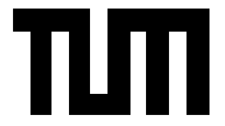
FAKULTÄT FÜR BAUINGENIEUR-
UND VERMESSUNGSWESEN

Hochgenaue numerische Lösung von Bewegungsproblemen mit frei wählbarer Stellengenauigkeit

Dissertation
von

Martin Ettl



TECHNISCHE UNIVERSITÄT
MÜNCHEN

Technische Universität München

Forschungseinrichtung für Satellitengeodäsie

Hochgenaue numerische Lösung von Bewegungsproblemen mit frei wählbarer Stellengenauigkeit

Martin Josef Ettl

Vollständiger Abdruck der von der Fakultät für Bauingenieur- und Vermessungswesen der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

genehmigten Dissertation.

Vorsitzender:		Univ.-Prof. Dr.-Ing. Uwe Stilla
Prüfer der Dissertation:	1.	Univ.-Prof. Dr. Urs Hugentobler
	2.	Univ.-Prof. Dr. Hans-Joachim Bungartz
	3.	Univ.-Prof. i.R. Dr. Manfred Schneider

Die Dissertation wurde am 06.07.2012 bei der Technischen Universität München eingereicht und durch die Fakultät für Bauingenieur- und Vermessungswesen am 16.10.2012 angenommen.

Inhaltsverzeichnis

Kapitel 1

Motivation

Angesichts der komplizierten Bewegungsprobleme kann sich die Bahn- und Parameterbestimmung in der Satellitengeodäsie und Himmelsmechanik meist nicht auf analytisch formulierte Lösungen stützen. Zudem wurde die Genauigkeit der Ortungs- und Bahnverfolgungsverfahren wie auch der satellitengestützten Messverfahren in den letzten Jahren beträchtlich gesteigert, was eine Verbesserung der numerischen Verfahren zur Verarbeitung der Daten unverzichtbar macht. Zwei Beispiele seien angeführt:

1. Die Satellitenmission GRACE zur Schwerefeldbestimmung, bei der Messungen des Abstandes der beiden Satelliten mit einer Genauigkeit von ca. $1.0\mu m$ und deren Relativgeschwindigkeiten mit etwa $1\frac{\mu m}{s}$ gemessen werden. Die Verarbeitung solcher Daten, die zudem in großer zeitlicher Dichte anfallen, führte auf die Frage, ob u.a. die Bahnen genügend genau dargestellt werden können. Abb.1.1 zeigt das Ergebnis einer numerischen Simulation der Bahnen, ausgeführt mit verschiedenen Integrationsverfahren. Es zeigte sich, dass das in der gängigen Vorgehensweise nicht ausreichend genau gelingt. Da GRACE-Nachfolgemissionen mit deutlich gesteigerter Messgenauigkeit[1] vorbereitet werden, stellt sich die Aufgabe einer hochgenauen Lösung des Bewegungsproblems umso dringlicher.

2. Laser-Entfernungsmessungen von Bodenstationen auf der Erde zu Reflektoreinheiten auf dem Mond gelingen seit mehreren Jahren mit etwa 1 cm Genauigkeit. Es besteht die Absicht, diese um eine Größenordnung durch verbesserte Auswertungsverfahren zu steigern. Bei der Verarbeitung der Messdaten muss die Bewegung des in das Sonnensystem eingebettete Erde-Mond-Systems in nachnewtonscher Näherung gelöst werden und zwar für den Zeitraum von über 30 Jahren, in dem Messungen vorliegen. Die Bestimmung von Modellparametern erfolgt nach dem Verfahren der differentiellen Korrektur von apriori-Werten, das iterativ durchgeführt werden muss. Dabei fallen bei jedem Schritt eine Ephemeridenrechnung des gesamten Sonnensystems, eine Berechnung der partiellen Ableitungen der testbaren Funktionale nach den Modellparametern sowie eine Ausgleichsrechnung der Beobachtungsgleichungen an. Die Iteration konvergiert nur, wenn die partiellen Ableitungen, die von der Genauigkeit der Ephemeridenrechnung abhängen, hinreichend genau gerechnet werden. Der Grund

[1] siehe mehr dazu unter [Flechtner12]

liegt darin, dass die Ausgleichsrechnung nach sehr flach ausgeprägten Minima suchen muss.

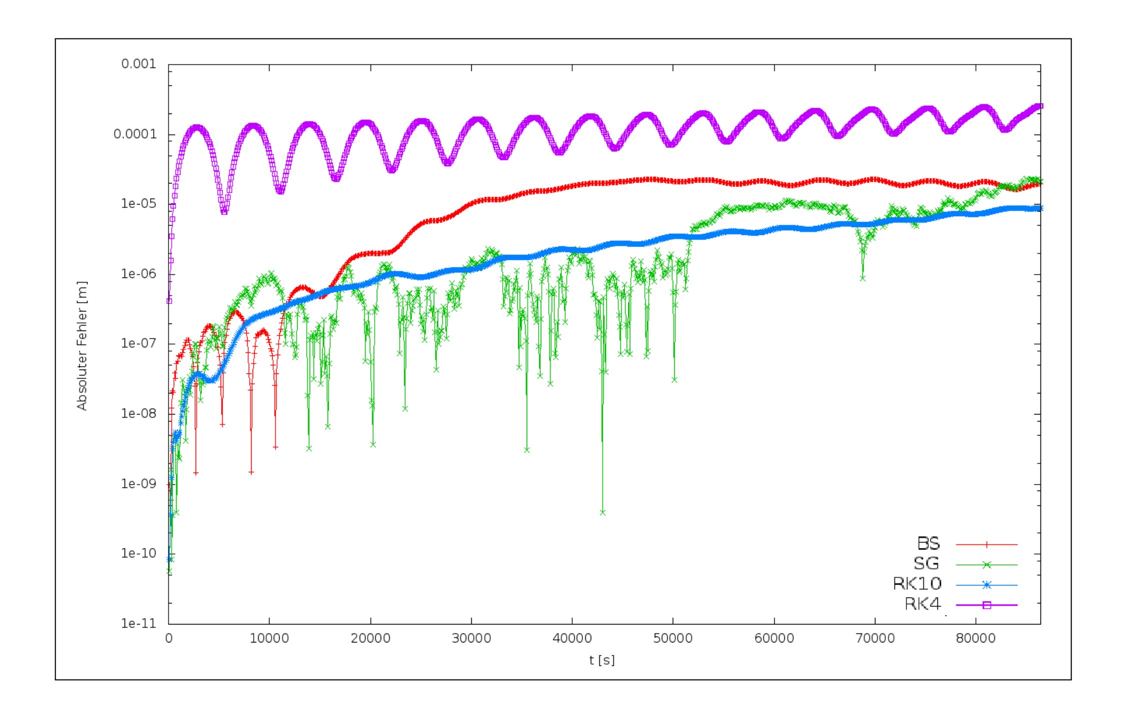


Abbildung 1.1: Abstandsdifferenzen verschiedener Lösungen über einen Simulationszeitraum von einem Tag (16 Umläufe)

Die vorgestellten Beispiele waren Anlass, numerische Verfahren für die in Zukunft steigenden Anforderungen bei der Auswertung der Messdaten bereitzustellen. Das wird noch dringlicher werden, wenn die in Entwicklung befindlichen optischen Uhren (Stabilität[2] bei etwa 10^{-17}) in den Messverfahren zum Einsatz kommen. Über die numerische Lösung von Bewegungsproblemen hinaus werden künftig alle Schritte der Datenvorverarbeitung sowie der Auflösung großer Gleichungssysteme auf ausreichende Genauigkeit zu überprüfen sein.
Die Lösung der anfallenden Bewegungsprobleme ist dabei eine zentrale Aufgabe, die interdisziplinär angegangen werden muss (Abb.1.2). Hierzu liefert die Numerische Mathematik das nötige theoretische Fundament zur Lösung gewöhnlicher Differentialgleichungen inklusive verlässlicher Standardverfahren. Sowohl die Implementierung der Lösungsfahren als auch die Rechnung mit *multiprecision*-Bibliotheken ist Bestandteil der Fachdisziplin Informatik. Die Lösung einer konkreten Aufgabenstellung, wie beispielsweise die Modellierung einer Satellitenbahn im Gravitationsfeld der Erde, erfordert Fachwissen aus dem Bereich der Himmelsmechanik und der Geodäsie. Bei der Umsetzung wurde darauf geachtet, dass auch verwandte Aufgabenstellungen anderer Fachdisziplinen gelöst werden können.

[2]siehe mehr dazu unter [Rosenband08]

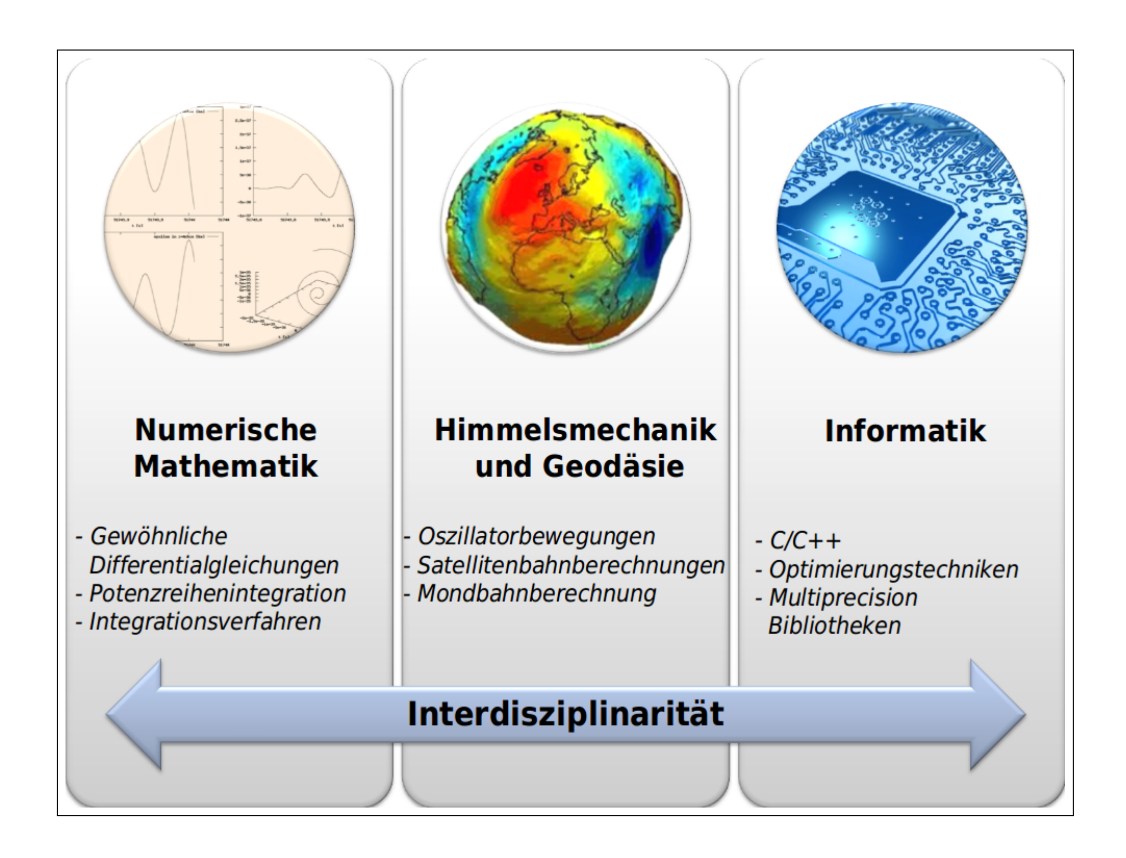


Abbildung 1.2: Interdisziplinäre Einordnung der Arbeit

Im folgenden Kapitel 2 wird zunächst der theoretische Hintergrund zur Lösung gewöhnlicher Differentialgleichungen und die dafür existierenden Lösungsverfahren vorgestellt sowie deren Eigenschaften anhand einschlägiger Beispiele untermauert. Daraufhin werden in Kapitel 3 die Aspekte der Implementierung jeweiliger Lösungsverfahren diskutiert. In Kapitel 4 werden einige *multiprecision*-Bibliotheken aufgrund ihres Funktionsumfangs und Laufzeitverhaltens evaluiert. Um allgemein unnötig lange Programmlaufzeiten zu verhindern, werden in Kapitel 5 unterschiedliche Techniken zur Verbesserung von Programmen vorgestellt. Darüber hinaus werden im Kapitel 6 verschiedene Methoden zur numerischen Differentiation präsentiert. In Kapitel 7 wird ein Konzept einer *Toolbox* zur Lösung gewöhnlicher Differentialgleichungen erarbeitet und mit Beispielen untermauert. Daraufhin werden in Kapitel 8 Anwendungsszenarien aus dem Bereich der Himmelsmechanik und Geodäsie gezeigt. Schließlich wird in Kapitel 9 ein Ausblick zu weiteren Entwicklungen und möglichen zukünftigen Anwendung geboten.

Kapitel 2

Lösung gewöhnlicher Differentialgleichungen

Schwerpunkt des Kapitels:

Viele technische und physikalische Vorgänge lassen sich durch gewöhnliche Differentialgleichungen beschreiben und stellen somit ein mathematisches Grundgerüst des jeweils untersuchten Problems dar. Dadurch wird beispielsweise die zeitliche und räumliche Veränderung eines Objekts inklusive aller Einflussgrößen und Störbeschleunigungen im Raum beschrieben. In diesem Abschnitt werden verschiedene Lösungsverfahren (Integratoren) gewöhnlicher Differentialgleichungen vorgestellt und deren theoretischer Hintergrund beleuchtet. Diese werden schließlich hinsichtlich ihrer Abarbeitungsgeschwindigkeit und ihres Fehlerhaushalts anhand praktischer Beispiele untersucht und bewertet.

2.1 Fehlernormen

Um die Güte von Ergebnissen anhand von Referenzlösungen im zeitlichen Verlauf beurteilen zu können, werden einige verschiedene Fehlernormen benötigt. Diese werden im Folgenden definiert, um später darauf zurückgreifen zu können.

2.1.1 Absoluter Fehler

Der absolute Fehler entspricht der tatsächlichen Abweichung zwischen gemessenen[1] Wert $Y(t_0)$ und dem wahren Wert $y(t_0)$ zum Zeitpunkt t_0. Damit kann der absolute Fehler $\epsilon_a(t_0)$ definiert werden durch:

$$\epsilon_a(t_0) = Y(t_0) - y(t_0) \tag{2.1.1}$$

In Abb. 2.1 ist dieser Zusammenhang illustriert. Dieses Fehlermaß trägt stets die Einheit des untersuchten Werts.

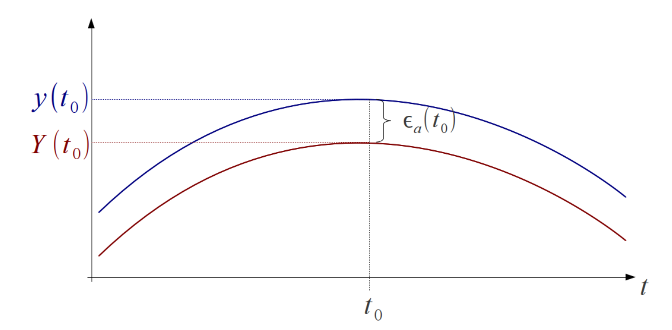


Abbildung 2.1: Absoluter Fehler

2.1.2 Relativer Fehler

Der relative Fehler $\epsilon_r(t_0)$ bezeichnet den prozentualen Anteil der Abweichung des gemessenen Werts $Y(t_0)$ vom wahren Wert $y(t_0)$ zum Zeitpunkt t_0. Dieser lässt sich formell beschreiben durch:

$$\epsilon_r(t_0) = \frac{\epsilon_a(t_0)}{y(t_0)} = \frac{Y(t_0) - y(t_0)}{y(t_0)} \tag{2.1.2}$$

2.1.3 Kumulativer Fehler

Beim kumulativen Fehler $\epsilon_k(t_i)$ werden die jeweils gemessenen Fehlergrößen $\epsilon(t_i)$ mit $i \in N$ aufsummiert. Dadurch steht zu jedem Zeitpunkt t_i das jeweilige Fehlerbudget fest. Somit kann das Anwachsen des Fehlers über der Zeit sichtbar gemacht werden und ist wie folgt definiert:

$$\epsilon_k(t_j) = \sum_{i=0}^{j} \epsilon(t_i) \tag{2.1.3}$$

[1] In dieser Arbeit wird nur in wenigen Fällen mit gemessenen Werten gearbeitet. Meistens werden berechnete bzw. simulierte Daten miteinander verglichen.

2.1.3.1 Kumulativer absoluter Fehler

Der kumulative absolute Fehler $\epsilon_{k_a}(t_i)$ bezeichnet die Fehlersumme des absoluten Fehlers ϵ_a bis zum Zeitpunkt t_i.

2.1.3.2 Kumulativer relativer Fehler

Der kumulative absolute Fehler $\epsilon_{k_r}(t_i)$ bezeichnet die Fehlersumme des relativen Fehlers ϵ_r bis zum Zeitpunkt t_i.

2.1.4 Gewichtete l_2-Norm

In vielen der folgenden Problemstellungen liegen die Ergebnisse in kartesischen Koordinaten x, y, z vor. Um die Qualität der jeweiligen Messergebnisse beurteilen zu können wäre es prinzipiell möglich, diese komponentenweise mit der analytischen Lösung zu vergleichen. Da dies aufgrund der vielen Vergleiche zu unübersichtlich werden würde, kann eine gewichtete l_2-Norm Θ gewählt werden. Diese liefert ein kompakteres Fehlermaß und ist wie folgt definiert:

$$\Theta = \sum_{i=0}^{k} \sqrt{\frac{(\epsilon_{a_x}^i)^2 + (\epsilon_{a_y}^i)^2 + (\epsilon_{a_z}^i)^2}{y(t_i)_x{}^2 + y(t_i)_y{}^2 + y(t_i)_z{}^2}}$$

mit

$$\begin{aligned} \epsilon_{a_x}^i &= Y(t_i)_x - y(t_i)_x \\ \epsilon_{a_y}^i &= Y(t_i)_y - y(t_i)_y \\ \epsilon_{a_z}^i &= Y(t_i)_z - y(t_i)_z \end{aligned} \tag{2.1.4}$$

Die l_2-Norm kann als relative Ortsabweichung betrachtet werden.

2.1.5 Mittlerer Fehler

Oft ist es hilfreich, einer Simulation eine Fehlerkennzahl zuordnen zu können. Dies kann mit Hilfe des mittleren Fehlers $\bar{\epsilon}$ bewerkstelligt werden, der wie folgt definiert ist:

$$\bar{\epsilon} = \frac{1}{m} \sum_{i=1}^{m} \epsilon(t_i) = \sqrt{\frac{1}{m} \epsilon_k(t_i)} \tag{2.1.5}$$

Wobei m die Anzahl der Fehlerterme darstellt und ϵ_k der kumulative Fehler ist. Der Fehlermittelwert könnte auch als mittlerer, kumulativer Fehler bezeichnet werden.

2.2 Fehlerarten

Nachdem im vorigen Abschnitt die Fehlermaße zur Beurteilung der Güte von Messwerten definiert wurden, ist es weiter erforderlich, den Gesamtfehler ϵ_g, welcher aus dem Vergleich zwischen Mess - und wahrem Wert resultiert, nach verschiedenen Fehlerarten aufzuschlüsseln. Prinzipiell kann der Gesamtfehler in folgende Teile zerlegt werden:

- *Eingangsfehler* $\epsilon_{eingang}$
  Bei dieser Fehlerart handelt es sich um einen kaum vermeidbaren Fehler, dessen Größenordnung aber bekannt sein sollte, da dieser die anfängliche Genauigkeit festlegt. Besonders bei der Integration von Satellitenbahnen sind die Anfangswerte nur mit einer bestimmten dezimalen Genauigkeit bekannt. Diese Unsicherheit wird als Eingangsfehler bezeichnet.

- *Verarbeitungsfehler bzw. Initialisierungsfehler*

  - *Rundungsfehler* ϵ_{rd}
    Der Rundungsfehler kann an verschiedener Stelle auftreten. Einerseits bei der Initialisierung einer Variablen mit einer Zahl, deren Wert nicht auf die interne Darstellung abgebildet werden kann (implizite Rundung). Andererseits bei der Verarbeitung zweier Gleitkommazahlen, bei denen am Ende der eigentlichen Operation (z.B. Addition) das Ergebnis auf die interne Darstellung abgebildet wird und ebenfalls gerundet wird.
  - *Diskretisierungsfehler (Verfahrensfehler)* ϵ_d
    Bei dieser Fehlerart wird das Ergebnis aufgrund der Güte des jeweils verwendeten Verfahrens mehr oder weniger verschlechtert. Falls beispielsweise die Auswertung einer Taylorreihe bei einer bestimmten Ordnung abgebrochen wird, dann entsteht ein lokaler Diskretisierungsfehler.

Es ergibt sich folgende formelle Aufgliederung des Gesamtfehlers:

$$\epsilon_g = \epsilon_{eingang} + \epsilon_{rd} + \epsilon_d \tag{2.2.6}$$

2.3 Numerische Integrationsverfahren

Die Bewegung eines Objektes kann durch Lösung von gewöhnlichen Differentialgleichungen dargestellt werden. Im einfachsten Fall handelt es sich hierbei um eine Differentialgleichung n-ter Ordnung. Diese können sowohl analytisch, als auch durch numerische Integration der Bewegungsgleichungen bestimmt werden. Derartige Lösungsverfahren werden eingesetzt, falls keine analytische Lösung vorhanden oder diese zu aufwendig zu lösen ist. Die zur numerischen Integration verwendeten Verfahren sind Standardverfahren aus der numerischen Mathematik zur Lösung von gewöhnlichen Differentialgleichungssystemen 1-ter und 2-ter Ordnung[2]. Anders als bei analytischen Lösungen wird hierbei keine zeitkontinuierliche Lösung erzeugt. Es wird lediglich zu diskreten Zeitpunkten durch Lösen eines Ersatzproblems[3] ein Resultat bestimmt. Mit anderen Worten, diese Verfahren erzeugen eine Wertetabelle mit Stützstellen, welche bestimmten Zeitpunkten zugeordnet sind. Wird eine Lösung zwischen den errechneten Stützstellen benötigt, so kann mittels Interpolation der jeweilige Zwischenwert bestimmt werden.

2.3.1 Gewöhnliche Differentialgleichungen

[DGL] Differentialgleichung

Eine gewöhnliche Differentialgleichung DGL ist eine Funktion, bei der neben der ge-

[2]siehe dazu [König03]
[3]siehe mehr dazu in [Schneider99], Seite 1043

suchten Funktion, auch Differentialquotienten der Funktion auftreten. Je nachdem, ob nur Ableitungen einer Funktion oder partielle Ableitungen mehrerer Variablen zugleich auftreten, wird zwischen gewöhnlichen und partiellen DGL unterschieden. Der Schwerpunkt dieser Arbeit liegt auf ausschließlich auf gewöhnlichen DGL.
Sei f eine stetige Funktion. So ist

$$f(x, y, y^{'}, y^{''}, ..., y^{(n)}) = 0 \tag{2.3.7}$$

ein gewöhnliches Differentialgleichungssystem mit n-ter Ordnung, wobei $y^{(n)}$ die n-te Ableitung ist. Im Allgemeinen besitzt dieses System unendlich viele Lösungen[4]. Eine spezielle Lösung wird durch die Wahl der Anfangsbedingungen festgelegt. Kann eine DGL außerdem nach der höchsten Ableitung aufgelöst werden, so wird diese als explizite DGL bezeichnet. Sie hat folgende Form:

$$y^{(n)} = f(x, y, y^{'}, y^{''}, ..., y^{(n-1)}) \tag{2.3.8}$$

Die hier untersuchten Lösungsverfahren können ausschließlich DGL 1-ter oder 2-ter Ordnung lösen. Liegt eine DGL höherer Ordnung vor, so kann diese stets auf ein System erster Ordnung zurückgeführt werden. Um den Einsatz gewöhnlicher Differentialgleichungen besser zu veranschaulichen, wird folgendes Beispiel aus [Huckle06] (Seite 280 und 281) entlehnt.

Beispiel 2.1 für „Freier Fall im konstanten Schwerefeld“

Ein Körper wird aus der Höhe h_0 zum Zeitpunkt $t_0 = 0$ losgelassen und unter dem Einfluss der konstanten Erdanziehung $-g$ nach unten beschleunigt. Zum Zeitpunkt des Loslassens ist die Anfangsgeschwindigkeit gerade $v_0 = 0$. Wir wollen sowohl die momentane Höhe $h(t)$ als auch die Geschwindigkeit $v(t)$ in Abhängigkeit von der Zeit t bestimmen.
Aus der Physik kennen wir die fundamentalen Beziehungen zwischen der Beschleunigung $a(t) = -g$, der Geschwindigkeit $v(t)$ und der Höhe $h(t)$ als System von gewöhnlichen Differentialgleichungen.

$$v(t) = \dot{h}(t) = \frac{dh}{dt} \tag{2.3.9}$$

$$a(t) = \dot{v}(t) = \frac{dv}{dt} = \frac{d^2h}{dt^2} = \ddot{h}(t) \tag{2.3.10}$$

Die Geschwindigkeit $v(t)$ bestimmen wir durch Integration von 2.3.9 mit der bekannten Beschleunigung $a(t) = -g$

$$v(t) = \int_{t_0}^{t} a(\tau)d\tau = -g\int_{0}^{t} d\tau = -gt \tag{2.3.11}$$

Die Geschwindigkeit $v(t)$ steigt somit proportional mit der verstrichenen Fallzeit t an. Die Proportionalitätskonstante entspricht der konstanten Beschleunigung $-g$. Um die Höhe $h(t)$ zu bestimmen, integrieren wir 2.3.9 und erhalten:

$$h(t) = h_0 + \int_{t_0}^{t} v(\tau)d\tau = h_0 - g\int_{0}^{t} \tau d\tau = h_0 - \frac{1}{2}gt^2. \tag{2.3.12}$$

[4] siehe dazu [Burlisch05]

Somit haben wir die Höhe $h(t)$ und die Geschwindigkeit $v(t)$ in Abhängigkeit der Zeit t berechnet und können z.B. die Aufschlagszeit t_1 und die entsprechende Aufschlagsgeschwindigkeit v_1 auf den Boden aus der Bedingung $h(t_1) = 0$ ermitteln:

$$t_1 = \sqrt{\frac{2h_0}{g}} \tag{2.3.13}$$

$$v_1 = -\sqrt{2h_0 g} \tag{2.3.14}$$

*Das aufgeführte Beispiel ist vergleichsweise einfach und durch zweifache Integration zu lösen, weil die auftretende Differentialgleichung nur eine Funktion der unabhängigen Variablen t und **nicht** von den abhängigen Variablen $h(t)$ oder $v(t)$ ist.*

Die Lösung einer Differentialgleichung besteht im Allgemeinen aus einer Vielzahl von Lösungen. Im gezeigten Beispiel wurde lediglich eine bestimme Lösung herausgegriffen, indem bestimmte Anfangswerte gewählt wurden.

2.3.2 Anfangswertprobleme [AWP]

[AB] Anfangsbedingung

Werden einer gewöhnlichen Differentialgleichung zusätzliche AB zugeordnet, so wird dies als Anfangswertproblem bezeichnet. Die Anfangsbedingungen $(y(t))$ sind wiederum einem bestimmten Zeitpunkt t zugeordnet und haben die Form:

$$\dot{y} = f(t, y) = f(t, y(t)) \tag{2.3.15}$$

mit den AB

$$y(t_0) = y_0 \tag{2.3.16}$$

Der Zeitraum, für den eine Lösung eines bestimmten AWPs berechnet wird, nennt man Integrationszeitraum.

2.3.3 Einteilung der numerischen Lösungsverfahren

Bei der numerischen Integration einer DGL wird eine Wertetabelle errechnet. Diese besteht aus Wertepaaren (x_i, Y_i), wobei $i \in I$ ist. Die berechneten Werte werden auch als Stützstellen bezeichnet und beschreiben beispielsweise die Bewegung eines Objekts im Raum. Berechnet das numerische Integrationsverfahren Stützstellen mit äquidistantem Abstand, so wird eine konstante Schrittweite ($h = const$) verwendet. Die Abszissen sind wie folgt definiert:

$$x_i = x_0 + ih \tag{2.3.17}$$

wobei $i = 0, 1, ..., n; i \in I$ mit $I = [x_0, x_n]$
und $h = x_{i+1} - x_i > 0$

Die numerisch bestimmten Wertepaare (x_i, Y_i) stellen lediglich eine Approximation des tatsächlichen Funktionswerts dar. Wie genau der approximierte Wert an den wahren Wert angenähert ist, hängt vom Integrationsverfahren, dem verwendeten Datentyp und der Größe des Eingangsfehlers ab. Aus diesem Grund muss Folgendes beachtet werden:

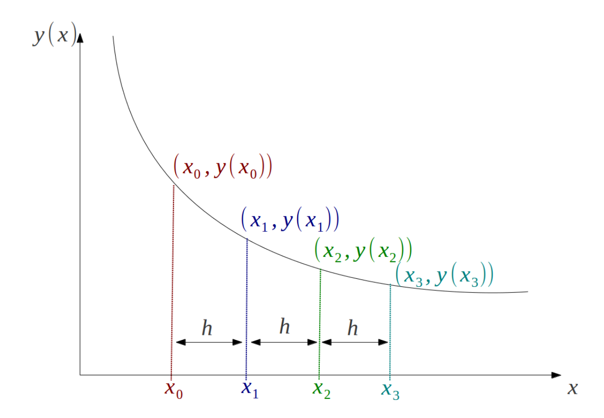


Abbildung 2.2: Ermittlung der Näherungslösung durch numerische Integration mit konstanter Schrittweite

$$\underbrace{Y_i := Y(x_i)}_{approximiert} \approx \underbrace{y(x_i) =: y_i}_{exakt}$$

Y_i steht für den numerisch angenäherten Funktionswert und y_i für den exakten Funktionswert an der Stelle x_i.

Durch formale Integration über $I = [x_0, x_n]$ ergibt sich folgende Integralgleichung:

$$\int_{x_0}^{x_n} \dot{y}(t)dt = y(x_n) - y(x_0) = \int_{x_0}^{x_n} f(t, y(t))dt \tag{2.3.18}$$

$$y(x) = y(x_0) + \int_{x_0}^{x_n} f(t, y(t))dt \tag{2.3.19}$$

Es soll die Stützstelle $x_1 \epsilon I$ des AWP berechnet werden. Wird somit ein einzelner Integrationsschritt betrachtet, so entsteht folgender Ausdruck:

$$y(x_1) = y(x_0) + \int_{x_0}^{x_1} f(t, y(t))dt \tag{2.3.20}$$

2.3.3.1 Explizite Lösungsverfahren

Bei den expliziten Verfahren werden zur Berechnung ausschließlich zeitlich zurückliegende Werte verwendet. Diese Verfahren finden beispielsweise im Prädiktor-Schritt der Prädiktor-Korrektor-Verfahren ihre Verwendung.

2.3.3.2 Implizite Lösungsverfahren

Die impliziten Lösungsverfahren verwenden sowohl zeitlich zurückliegende, als auch vorhergesagte Werte. Diese Verfahren werden als Korrektor in Prädiktor-Korrektor-Verfahren eingesetzt.

2.3.3.3 Einschrittverfahren

Die Einschrittverfahren benötigen zur Berechnung eines weiteren approximierten Wertes Y_{i+1} ausschließlich einen vorherigen Wert Y_i. Weitere vorherige Werte Y_{i-z} wer-

den dabei nicht in Betracht gezogen. Ein Einschrittverfahren wird durch folgende Rekursion beschrieben:

$$Y(x_{i+1}) = y(x_i) + h\Phi(x_i, y(x_i), h, f) \tag{2.3.21}$$

Die Funktion Φ wird als Inkrementfunktion bezeichnet. Ist die Inkrementfunktion unabhängig von der Schrittweite h, so ist es ein explizites Einschrittverfahren. Ist die Inkrementfunktion von h abhängig ($\Phi(h)$), so ist das Verfahren implizit. Bekannte Einschrittverfahren sind das klassische Runge-Kutta-Verfahren, das Euler-Cauchy-Verfahren und das Verfahren von Heun. Hierbei wird eine Taylorreihe mit einer bestimmten Ordnung zur Extrapolation verwendet.

2.3.3.4 Mehrschrittverfahren

Bei den Mehrschrittverfahren werden zur Berechnung eines Funktionswerts Y_{i+1} eine Menge $z \geq 1$ vorangegangener Funktionswerte $Y_{i-z}, Y_{i-z+1}, ..., Y_{i-1}, Y_i$ in Betracht gezogen. Bekannte Vertreter aus der Klasse der Mehrschrittverfahren stellen die Integrationsverfahren von Shampine-Gordon, Störmer-Cowell und Burlisch-Stör dar. Diese Verfahren benötigen zur Berechnung gleichmäßig verteilte Zeitschritte. Pro Integrationsschritt sind zwei Aufrufe der DGL nötig. Der erste Schritt wird als **Prädiktor-Schritt** und der zweite als **Korrektor-Schritt** bezeichnet.

[DGL] Differentialgleichung

Der **Prädiktor-Schritt** hat folgende Form[5]:

$$y_{i+1} = y_i + h \sum_{m=1}^{n} \alpha_m f(h * x_{i+1-m}, y_{i+1-m}) \tag{2.3.22}$$

Die α_m-Koeffizienten des Prädiktor-Schritts werden wie folgt ermittelt:

$$\begin{aligned} f(x) &= a_0 + a_1 x + a_2 x^2 + ... + a_n x^n \\ f'(x) &= a_1 + 2a_2 x + ... + n a_n x^{n-1} \end{aligned} \tag{2.3.23}$$

Wird f für $x_0 = 0$ und $x_1 = 1$ ausgewertet, so ist

$$\begin{aligned} f(0) &= a_0 \\ f(1) &= a_0 + a_1 + ... + a_n \end{aligned} \tag{2.3.24}$$

Somit ist $f(1) = f(0) + df$ mit $df = a_1 + ... + a_n$. Für die zurückliegenden Zeitpunkte gilt Folgendes:

$$\begin{aligned} f'(0) &= a_1 \\ f'(1) &= a_1 - 2a_2 + 3a_3 + ... \\ f'(2) &= a_1 - 4a_2 + 12a_3 + ... \\ &... \\ f'(-m) &= a_1 + 2a_2(-m) + 3a_3(-m)^2 + ... + n a_n (-m)^{n-1} \end{aligned} \tag{2.3.25}$$

Werden die Koeffizienten in Form einer Matrixschreibweise dargestellt, so gilt

$$\begin{aligned} f' &= Ma \\ a &= M^{-1} f' \end{aligned} \tag{2.3.26}$$

[5] siehe dazu auch [Gruber10], Seite 61

Somit wird df aus der Summe der Koeffizienten von a gebildet.

$$df = a^T I = (f')^T \underbrace{(M^T)^{-1} I}_{\alpha_m} \tag{2.3.27}$$

Die α_m - Koeffizienten werden durch $(M^T)^{-1}I$ bestimmt. Der **Korrektor-Schritt** verwendet den vorhergesagten Zustand, um die Gleichung auszuwerten und zu verbessern.

$$y_{i+1} = y_i + h \sum_{m=0}^{n} \beta_m f(h * x_{i+1-m}, y_{i+1-m}) \tag{2.3.28}$$

Dabei enthält β_m die Koeffizienten der Korrekturwerte. Die Vorgehensweise ist dabei analog zu der des Prädiktor-Schritts. Es wird lediglich der Zeitpunkt x_{i+1} mit einbezogen.

2.3.3.5 Erweiterung der Prädiktor-Korrektor-Verfahren mit adaptiver Schrittweitensteuerung [SWS]

Ein Prädiktor-Korrektor-Verfahren kann zusätzlich mit einer adaptiven Schrittweitensteuerung ausgestattet werden. Viele der gängigen Standardverfahren sind mit dieser Funktionalität ausgestattet. Dabei wird für jeden Integrationsschritt die Schrittweite so angepasst, dass der Diskretisierungsfehler möglichst minimal wird (siehe dazu 2.6.1 auf Seite 44). Dabei ist zu beachten, dass bei allen Verfahren lediglich der lokale Diskretisierungsfehler ϵ_l in Betracht gezogen wird und **nicht** der Gesamtfehler ϵ_g.

[SWS] Schrittweitensteuerung

2.3.4 Theorie der numerischen Lösungsverfahren

In den folgenden Unterpunkten wird auf die Lösungsverfahren eingegangen, welche im Rahmen dieser Arbeit verwendet wurden. Dabei werden zuerst die klassischen Runge-Kutta-Verfahren, dann verallgemeinerte Runge-Kutta-Verfahren und schließlich Verfahren mit adaptiver Schrittweitensteuerung betrachtet. Ferner wird auch auf alternative Ansätze, wie den Potenzreihenintegrator und die Taylorreihenmethode eingegangen. Zuletzt wird die Quadratur als weitere alternative Lösungsmethode vorgestellt.

2.3.4.1 Das Runge-Kutta-Verfahren 2.Ordnung [RK2]

Das Runge-Kutta-Verfahren 2.Ordnung ist ein Einschrittverfahren ohne automatische Schrittweitensteuerung. Ist der Funktionswert $y(x_i) = y_i$ als AB gegeben, so kann $Y(x_{i+1}) = Y_{i+1}$ durch Lösung folgender Gleichung mit Hilfe des Runge-Kutta-Verfahrens bestimmt werden:

[RK2] Runge-Kutta 2. Ordnung

$$Y_{i+1} = y_i + hk_2 + O(h^3) \tag{2.3.29}$$

wobei

$$\begin{aligned} k_1 &= hf(x_i, y_i) \\ k_2 &= hf(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}) \end{aligned} \tag{2.3.30}$$

Hierbei ist h die gewählte Schrittweite und $O(h^3)$ entspricht der Konsistenzordnung. Für einen Integrationsschritt benötigt dieses Verfahren 2 Auswertungen der Differentialgleichung, 3 Additionen, 3 Multiplikationen und 2 Divisionen.

2.3.4.2 Das Runge-Kutta-Verfahren 3.Ordnung [RK3]

[RK3] Runge Kutta 3. Ordnung

Das Runge-Kutta-Verfahren 3.Ordnung ist ein Einschrittverfahren ohne automatische Schrittweitensteuerung. Ist der Funktionswert $y(x_i) = y_i$ als AB gegeben, so kann $Y(x_{i+1}) = Y_{i+1}$ durch Lösung folgender Gleichung mit Hilfe des Runge-Kutta-Verfahrens 3.Ordnung bestimmt werden:

$$Y_{i+1} = y_i + \frac{h}{6}(k_1 + 4k_2 + k_3) + O(h^4) \tag{2.3.31}$$

wobei

$$\begin{aligned} k_1 &= hf(x_i, y_i) \\ k_2 &= hf(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}) \\ k_3 &= hf(x_i + h, y_i + k_2) \end{aligned} \tag{2.3.32}$$

Hierbei ist h die gewählte Schrittweite und $O(h^4)$ entspricht der Konsistenzordnung. Für einen Integrationsschritt benötigt dieses Verfahren 3 Auswertungen der Differentialgleichung, 7 Additionen, 4 Multiplikationen und 3 Divisionen.

2.3.4.3 Das Runge-Kutta-Verfahren 4.Ordnung [RK4]

[RK4] Runge Kutta 4. Ordnung

Das Runge-Kutta-Verfahren 4.Ordnung ist ein Einschrittverfahren ohne automatische Schrittweitensteuerung. Ist der Funktionswert $y(x_i) = y_i$ als AB gegeben, so kann $Y(x_{i+1}) = Y_{i+1}$ mit dem Runge-Kutta-Verfahren bestimmt werden:

$$Y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + O(h^5) \tag{2.3.33}$$

$$\begin{aligned} k_1 &= hf(x_i, y_i) \\ k_2 &= hf(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}) \\ k_3 &= hf(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}) \\ k_4 &= hf(x_i + h, y_i + k_3) \end{aligned} \tag{2.3.34}$$

Dabei ist h die Integrationsschrittweite und $O(h^5)$ stellt die Konsistenzordnung des Verfahrens dar. Für einen Integrationsschritt benötigt dieses Verfahren 4 Auswertungen der Differentialgleichung, 10 Additionen, 7 Multiplikationen und 5 Divisionen.

2.3.4.4 Das Runge-Kutta-Verfahren N.Ordnung [RKN]

Wie aus den bereits erwähnten RK-Verfahren unterschiedlicher Ordnung ersichtlich wird, gibt es eine allgemeine, formale Darstellung für explizite Runge-Kutta-Verfahren N.Ordnung. Diese hat folgende Gestalt:

$$y_{i+1} = y_i + h \sum_{r=1}^{R} c_r k_r \tag{2.3.35}$$

mit den jeweiligen k_r

$$\begin{aligned} k_1 &= f(x_i, y_i) \\ k_2 &= f(x_i + ha_2, y_i + hb_{21}k_1) \\ &\dots \\ k_n &= f(x_i + ha_n, y_i + h\sum_{s=1}^{n-1} b_{ns}k_s) \end{aligned} \tag{2.3.36}$$

Die Gleichung 2.3.35 gibt dabei die Annäherung eines Integrationsschritts an. Die Parameter eines RK-Verfahrens einer bestimmten Ordnung werden ermittelt, indem ihre formelmäßige Darstellung einer Taylor-Approximation derselben Ordnung gegenübergestellt wird.

$$\underbrace{\sum_{r=1}^{R} \frac{h^{r-1}}{r!} f^{(r-1)}(x, y(x)) + O(h^n)}_{Taylor-Approximation} \neq \underbrace{\sum_{r=1}^{R} c_r k_r}_{RK-Verfahren} \tag{2.3.37}$$

Die freien Parameter b_{ns}, a_i und c_r können aus dem Vergleich der Runge-Kutta-Formel mit der jeweiligen Taylor-Approximation konstruiert werden. Bei dieser Vorgehensweise kann es vorkommen, dass einige Parameter unbestimmt bleiben. Aus diesem Grund gibt es in der Literatur für bestimmte Ordnungen unterschiedliche Sätze von Parametern. Es gibt prinzipiell zwei verschiedene Arten von Runge-Kutta-Methoden. Die expliziten Verfahren, welche nur von den Koeffizienten $k_1, ..., k_n$ abhängen und die impliziten Verfahren, die nur iterativ gelöst werden können. Um die hier dargestellte allgemeine Form der Parameterbestimmung besser zu veranschaulichen, sollen an folgendem Beispiel die Parameter des expliziten RK2-Verfahrens berechnet werden.

Beispiel 2.2 „Bestimmung der freien Parameter des expliziten RK2-Verfahrens“
Im ersten Schritt soll die RK-Formel für den Fall N = 2 ausgearbeitet werden.

$$\begin{aligned} c_1k_1 + c_2k_2 &= c_1 f(x_i, y_i) + c_2 f(x_i + ha_2, y_i + hb_{21} f(x_i, y_i)) \\ &= c_1 f(x_i, y_i) + c_2[f(x_i, y_i) + a_2 h f(x_i, y_i) + b_{21} h f(x_i, y_i)\partial_y f(x_i, y_i)] \\ &= (c_1 + c_2) f(x_i, y_i) + h[c_2 a_2 \partial_x f(x_i, y_i) + c_2 b_{21} f(x_i, y_i)\partial_y f(x_i, y_i)] \end{aligned} \tag{2.3.38}$$

Hierfür wurde $k_1 = f(x_i, y_i)$,$k_2 = f(x_i+ha_2, y_i+hb_{21}k_1)$ *und* $\frac{dy(x)}{dx} = f(x_i, y_i)$ *verwendet. Zum Vergleich wird noch die Taylor-Entwicklung benötigt. Diese hat folgende Form:*

$$\sum_{r=1}^{R} \frac{h^{r-1}}{r!} f^{(r-1)}(x, y(x)) = f(x, y) + \frac{h}{2}[\partial_x f(x, y) + f(x, y)\partial_y f(x, y)] + O(h^2) \tag{2.3.39}$$

Werden nun die beiden erarbeiteten Lösungen untereinander geschrieben, so können die freien Parameter bestimmt werden.

$$(c_1 + c_2) f(x_i, y_i) \quad +h[c_2 a_2 \quad \partial_x f(x_i, y_i) + c_2 b_{21} \quad f(x_i, y_i)\partial_y f(x_i, y_i)] \tag{2.3.40}$$

$$f(x, y) \quad +\frac{h}{2}[\quad \partial_x f(x, y)+ \quad f(x, y)\partial_y f(x, y)] \tag{2.3.41}$$

Aus dem Vergleich können Werte für die Parameterkombinationen abgelesen werden:

$$\begin{aligned} c_1 + c_2 &= 1 \\ c_2 a_2 = c_2 b_{21} &= \frac{1}{2} \end{aligned} \tag{2.3.42}$$

Daraus können die Werte der freien Parameter bestimmt werden: $c_1 = 0$, $c_2 = 1$, $a_2 = b_{21} = \frac{1}{2}$. *Werden diese Parameter verwendet, dann ergibt dies folgende formelmäßige Darstellung:*

$$\begin{aligned} k_1 &= f(x_i, y_i) \\ k_2 &= f(x_i + \frac{h}{2}, y_i + \frac{h}{2} k_1) \\ Y_{i+1} &= y_i + hk_2 \end{aligned} \tag{2.3.43}$$

Die formelmäßige Darstellung entspricht exakt der Definition des RK2-Verfahrens.

Im Rahmen dieser Arbeit wurden außerdem das explizite RK-Verfahren 8.- und 10.Ordnung implementiert. Deren formelmäßige Darstellung unterliegt derselben Theorie wie die Verfahren niedriger Ordnung, welche bereits erwähnt wurden. Sie besitzen jedoch eine höhere Fehlerordnung und benötigen entsprechend ihrer umfangreicheren Implementierung mehr Rechenoperationen für einen Integrationsschritt.

2.3.4.5 Das Runge-Kutta-Fehlberg-Verfahren [RKF]

Das Runge-Kutta-Fehlberg-Verfahren einer bestimmten Ordnung ist, verglichen mit klassischen RK-Verfahren bei gleicher Ordnung, genauer. Dieser Zuwachs an Genauigkeit wird mit etwas höherem Rechenaufwand erkauft[6]. Hierbei wird zur Lösung der Differentialgleichung örtliche Steigungsinformation ausgenutzt, was zu einer Verbesserung der lokalen Fehlerordnung führt. Ferner kann hier mit relativ geringem Aufwand eine adaptive Schrittweitensteuerung eingebaut werden. Die Formeln zur Be-

[6] siehe dazu [Jordan72], ab Seite 309

rechnung eines Integrationsschritts mit der expliziten RKF (4/5.Ordnung) haben folgende Gestalt:

$$Y_{i+1} = y_i + \frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4104}k_4 - \frac{1}{5}k_5 \tag{2.3.44}$$

wobei

$$\begin{aligned}
k_1 &= hf(x_i, y_i) \\
k_2 &= hf(x_i + \frac{1}{4}h, y_i + \frac{1}{4}k_1) \\
k_3 &= hf(x_i + \frac{3}{8}h, y_i + \frac{3}{32}k_1 + \frac{9}{32}k_2) \\
k_4 &= hf(x_i + \frac{12}{13}h, y_i + \frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3) \\
k_5 &= hf(x_i + h, y_i + \frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4) \\
k_6 &= hf(x_i + \frac{1}{2}h, y_i - \frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5)
\end{aligned} \tag{2.3.45}$$

Hier wurde ein zusätzlicher Term ($k6$) berechnet, mit dem ein verbessertes Ergebnis erzielt werden kann. Die Formel, welche den zusätzlichen k-Term berücksichtigt, hat folgendes Aussehen:

$$Y^*_{i+1} = y_i + \frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6 \tag{2.3.46}$$

Nun können die beiden Funktionswerte Y_{i+1} und Y^*_{i+1} berechnet werden, wobei letzterer genauer ist. Dies wird ausgenutzt um die Schrittweite h zu bestimmen. Dafür wird außerdem eine Fehlerschranke benötigt, die üblicherweise beim Start der Berechnung angegeben und hier mit ϵ bezeichnet wird. Daraus berechnet sich die Schrittweite wie folgt:

$$h = h(\frac{\epsilon h}{2|Y^*_{i+1} - Y_{i+1}|})^{\frac{1}{4}} \qquad \forall (Y^*_{i+1} - Y_{i+1}) \neq 0 \wedge h \leq h_{max} \tag{2.3.47}$$

wobei h_max die maximal erlaubte Schrittweite bezeichnet. Diese kann ebenfalls beim Start der Berechnung angegeben werden. Insgesamt benötigt dieses Verfahren für einen Berechnungsschritt, inklusive Bestimmung der Schrittweite, 6 Aufrufe der Differentialgleichung, 30 Additionen, 37 Multiplikationen, 27 Divisionen und eine Auswertung der vierten Wurzel. Das RKF-Verfahren ist detailliert in [Jordan72](ab Seite 309) beschrieben. Dort findet sich auch ein Hinweis zur Erweiterung des RKF-Verfahrens für höhere Ordnungen. Dieses Verfahren wurde nicht im Rahmen dieser Arbeit implementiert.

2.3.4.6 Die Modified Midpoint Methode [MMID]

Die MMID ist ein Einschrittverfahren 2.Ordnung ohne adaptive Schrittweitensteuerung. Prinzipiell kann dieses Verfahren als eigenständiges Integrationsverfahren angesehen werden. In der Praxis wird es allerdings als Hilfsmethode innerhalb des Burlisch-Stoer-Integrators verwendet. Hierbei wird ein bestimmter Integrationsschritt der Schrittweite h in n äquidistante Zwischenschritte $\tilde{h}$ unterteilt.

$$\tilde{h} = \frac{h}{n} \tag{2.3.48}$$

Es werden stets $n + 1$ Funktionsaufrufe pro Integrationsschritt benötigt. Mit dem Anfangswert $y(x)$, den jeweiligen Zwischenwerten z_i und dem Aufruf der rechten Seite $f(x, z_m)$ kann eine Berechnungsvorschrift für diese Methode angegeben werden:

$$\begin{aligned} z_0 &= y(x) \\ z_1 &= z_0 + \tilde{h} f(x, z_0) \\ z_2 &= ... \\ z_{m+1} &= z_{m-1} + 2\tilde{h} f(x + m\tilde{h}, z_m) \end{aligned} \tag{2.3.49}$$

Die Berechnung der approximierten Lösung $Y(x+h)$ an der Stelle $x+h$ erfolgt durch

$$Y(x + h) \approx Y_n = \frac{1}{2}(z_n + z_{n-1} + \tilde{h} f(x + h, z_n)), \tag{2.3.50}$$

wobei für $m = 1, 2, ..., n-1$ gilt. Diese Methode hat eine hilfreiche Eigenschaft: Wird die Fehlerfunktion in Form einer Potenzreihe in Abhängigkeit von h ausgedrückt, so kann gezeigt werden, dass diese nur gerade Potenzen enthält. Werden nun Schritte geschickt zusammengefasst, sodass ein Term höherer Ordnung wegfällt, dann kann die Genauigkeit gleich um zwei Größenordnungen zunehmen. In [Press07] (Seite 923) wird ein Beispiel[7] angegeben, das hier aufgegriffen wird.

Beispiel 2.3 „Fehlerabschätzung zur Modified Midpoint Methode“
For example, suppose n is even, and let $y_{n/2}$ denote the result of applying ... with half as many steps, $n \to n/2$. Then the estimate

$$Y(x + h) \approx \frac{4y_n - y_{n/2}}{3} \tag{2.3.51}$$

is fourth-order accurate, the same as fourth-order Runge-Kutta, but requires only about 1.5 derivative evaluations per step h instead of Runge-Kutta's four evaluations.

Weitere Informationen zur Fehlerabschätzung sind in [Press07] ab Seite 923 und in [Press92] ab Seite 722 zu finden.

2.3.5 Symplektische Integrationsverfahren [SI]

Die Symplektischen Integrationsverfahren sind eine Untergruppe der geometrischen Lösungsverfahren für gewöhnliche Differentialgleichungen. Diese spezielle Form der

[7] Teilweise wurden die Formelbuchstaben angepasst.

numerischen Integration erreicht die Konstanz von Erhaltungsgrößen wie z. B. Energie, besser als beispielsweise ein Runge-Kutta-Verfahren. Die SI-Verfahren wurden zur Lösung kanonischer Bewegungsgleichungen konzipiert, wobei:

$$\begin{aligned} \dot{p} &= -\frac{\partial H}{\partial q} \\ \dot{q} &= \frac{\partial H}{\partial q} \end{aligned} \tag{2.3.52}$$

Dabei bezeichnen q die generalisierten Koordinaten, p die generalisierten Geschwindigkeiten und H die Hamilton-Funktion[8]. Ein Satz aus Position und Geschwindigkeitswerten (p, q) werden ferner als kanonische Koordinaten bezeichnet. Information hierzu sind bei [Schneider93] (ab Seite 469) und in [Yoshida90] zu finden. Hierin wird die Konstruktion von symplektischen Integrationsverfahren hoher Ordnung diskutiert und Formeln sowie Koeffizienten für die Ordnungen 4, 6 und 8 angegeben. Diese wurden mit Hilfe der Baker-Campbell-Hausdorff[9] Formel erarbeitet und versprechen eine schnelle rechnertechnische Abarbeitung. Um die nötigen Koeffizienten zu bestimmen wurde ferner von [Yoshida90] ein iteratives Verfahren zur Bestimmung der benötigten Nullstellen eingesetzt (*Brents-method*[10]). Mit Hilfe dieser Methode wurden für die 6.Ordnung drei verschiedene Parametersätze (für die 8.Ordnung fünf) bestimmt. Die numerischen Werte der Parameter w_1, w_2, w_3 der 6.Ordnung sind in Tabelle 2.1 angegeben. Wobei die verschiedenen Lösungsvarianten mit A, B und C bezeichnet werden.

Parameter	A	B	C
w_1	-0.117767998417887E1	-0.213228522200144E1	0.152886228424922E-2
w_2	0.235573213359357	0.426068187079180E-2	-0.214403531630539E1
w_3	0.784513610477560	0.14398481679767E1	0.144778256239930E1

Tabelle 2.1: Parameter des symplektischen Integrators 6.Ordnung

[8] siehe mehr dazu unter [Stephani95], Seite 147

[9] siehe mehr dazu unter [Reinsch00]

[10] siehe mehr dazu unter [Press92] auf Seite 454, Kapitel 9. *Root Finding and Nonlinear Sets of Equations*

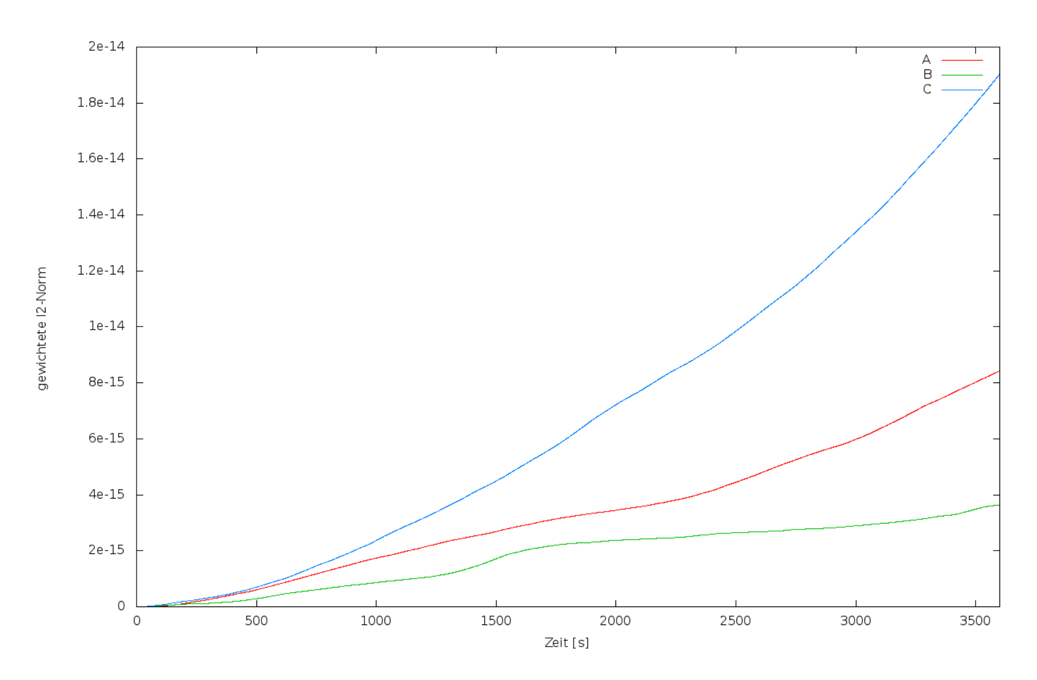


Abbildung 2.3: Verifikation verschiedener Parametersätze für den Symplektischen Integrator 6.Ordnung über einen Zeitraum von einer Stunde

Beispiel 2.4 „Überprüfung der Stabilität bei Verwendung verschiedener Parametersätze zur Integration 6.Ordnung“

Um herauszufinden welcher Parametersatz die genaueste Lösung liefert, soll hier folgende Aufgabenstellung mit jedem der Parametersätze A, B und C aus Tabelle 2.1 gelöst und einander gegenübergestellt werden. Als zu lösendes Problem wird das Keplerproblem[11] verwendet, da dafür eine analytische Lösung vorhanden ist. Es ist somit zur Verifikation der numerisch bestimmten Lösung geeignet.
Um die Kurzzeitstabilität der jeweiligen Lösung beurteilen zu können, wurden drei Lösungen einer Keplerbahn über einen Zeitraum von einer Stunde (3600 Sekunden) berechnet. Die jeweiligen Lösungen sind im kartesischen Koordinatensystem (x, y, z) *angegeben und könnten prinzipiell gegen eine analytische Lösung verglichen werden. Da dies aufgrund der vielen Vergleiche zu unübersichtlich erscheint, wurde eine gewichtete* l_2*-Norm* Θ *gewählt (siehe dazu in Abschnitt 2.1.4 auf Seite 7). Hierbei steht* k *für die Anzahl der berechneten Ergebnisse.* $Y(t_i)_x$ *ist das numerisch berechnete Ergebnis zum Zeitpunkt* t_i *der x-Komponente. Außerdem ist* $y(t_i)_x$ *das analytisch berechnete Ergebnis zum Zeitpunkt* t_i *der x-Komponente. Zur Berechnung der Lösungen A, B und C wurden jeweils 3600 Ergebnisse berechnet (mit* $k = 3600$*). Das Ergebnis des Vergleichs ist in Abbildung 2.3 dargestellt. Bemerkenswert ist hier der signifikante Anstieg der* Θ*-Norm bei der Lösung des Parametersatzes C über einen relativ kurzen Berechnungszeitraum von einer Stunde. In diesem Fall ist es zu empfehlen, den Parametersatz B für die Berechnung zu wählen, da dieser den kleinsten Anstieg verursacht. Wird der Zeitraum von einer Stunde auf einen Tag erweitert (*$k = 86400$*), dann ändert sich die Situation. Wird über einen längeren Zeitraum integriert, so ist der Parametersatz C der Variante A vorzuziehen. Zusammenfassend ist der Parametersatz B für kurze Integrationszeiten empfehlenswert und C für längere Zeitintervalle.*

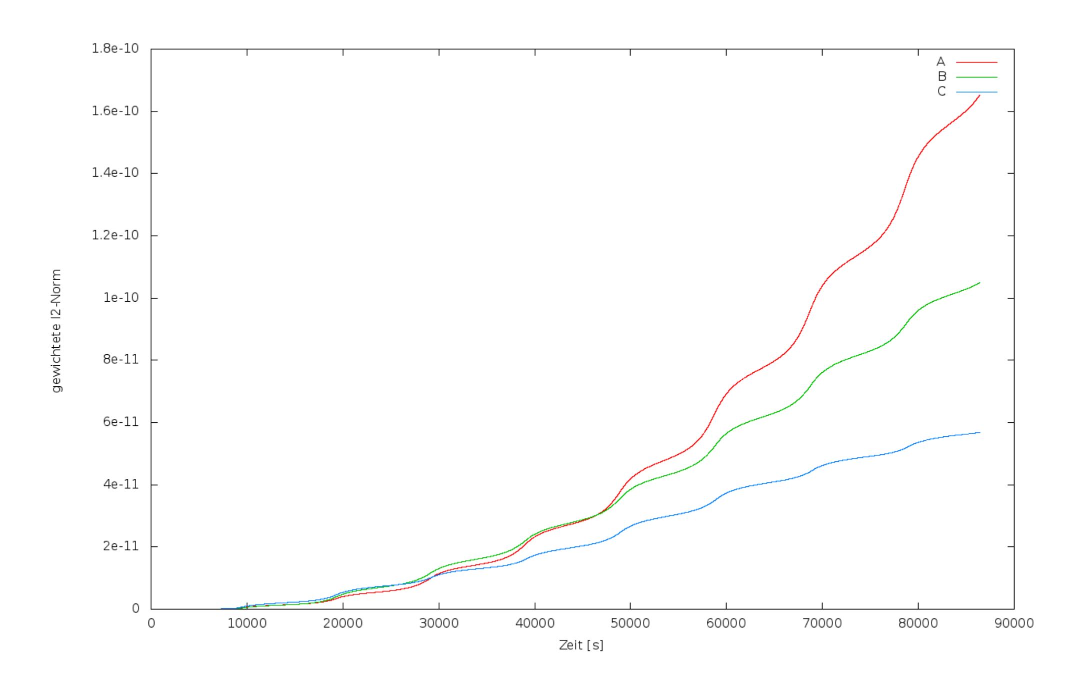


Abbildung 2.4: Verifikation verschiedener Parametersätze für den symplektischen Integrator 6.Ordnung über einen Zeitraum von einem Tag

Die Ergebnisse dieser Untersuchung fordern geradezu eine genauere Betrachtung der Parameter 8.Ordnung hinsichtlich Kurz - und Langzeitverhalten.

Beispiel 2.5 „Überprüfung der Stabilität bei Verwendung verschiedener Parametersätze zur Integration 8.Ordnung“
Im Gegensatz zur 6.Ordnung wurden von [Yoshida90] für die 8.Ordnung fünf verschiedene Parametersätze zu je sieben Parametern ($w_1, ..., w_7$) erarbeitet. Diese sind in den Tabellen 2.2 und 2.3 aufgelistet. Analog zum vorigen Beispiel werden diese mit den Buchstaben A, B, C, D und E bezeichnet. In diesem Beispiel wird dieselbe Aufgabenstellung verwendet, wie im vorausgegangenen, d.h. es wurde für jeden der fünf Parametersätze eine Keplerbahn berechnet und das Ergebnis mit der analytischen Lösung verglichen. Um die Kurz - und Langzeitstabilität beurteilen zu können, wurde für ersteres wieder ein Integrationszeitraum von einer Stunde (3600 Sekunden) gewählt. Zur Verifikation der Langzeitstabilität wurde über einen Tag integriert (86400 Sekunden). Betrachtet man die Ergebnisse der Simulation (Abb. 2.5), welche über einen Zeitraum von 3600 Sekunden durchgeführt wurden, so liefert der Parametersatz D das schlechteste Ergebnis, wohingegen der Parametersatz B im zweiten Teil der Simulation besser zu sein scheint. Die anderen Parametersätze liegen dazwischen und sind von mehr oder weniger guter Qualität. Wird nun die zweite, etwas längere Simulation betrachtet (Abb. 2.6), so bestätigt sich das Ergebnis aus Abb. 2.5. Die Parametersätze B und D liefern auch über diesen Zeitraum ein schlechteres Ergebnis. Mit Abstand der bessere Parametersatz ist A, wobei C und E im Mittelfeld liegen.

Parameter	A	B	C
w_1	-0.161582374150097E+1	-0.169248587770116E-2	0.311790812418427E+0
w_2	-0.244699182370524E+1	0.289195744315849E+1	-0.155946803821447E+1

w_3	-0.716989419708120E-2	0.378039588360192E-2	-0.167896928259640E+1
w_4	0.244002732616735E+1	-0.289688250328827E+1	0.166335809963315E+1
w_5	0.157739928123617E+0	0.289105148970595E+1	-0.106458714789183E+1
w_6	0.182020630970714E+1	-0.233864815101035E+1	0.136934946416871E+1
w_7	0.104242620869991E+1	0.148819229202922E+1	0.629030650210433E+0

Tabelle 2.2: Parameter des symplektischen Integrators 8.Ordnung

Parameter	D	E
w_1	0.102799849391985E+0	0.227738840094906E-1
w_2	-0.196061023297549E+1	0.252778927322839E+1
w_3	0.193813913762276E+1	-0.719180053552772E-1
w_4	-0.158240635368243E+0	0.536018921307285E-2
w_5	-0.144485223686048E+1	-0.204809795887393E+1
w_6	0.253693336566229E+0	0.107990467703699E+0
w_7	0.914844246229740E+0	0.130300165760014E+1

Tabelle 2.3: Parameter des symplektischen Integrators 8.Ordnung

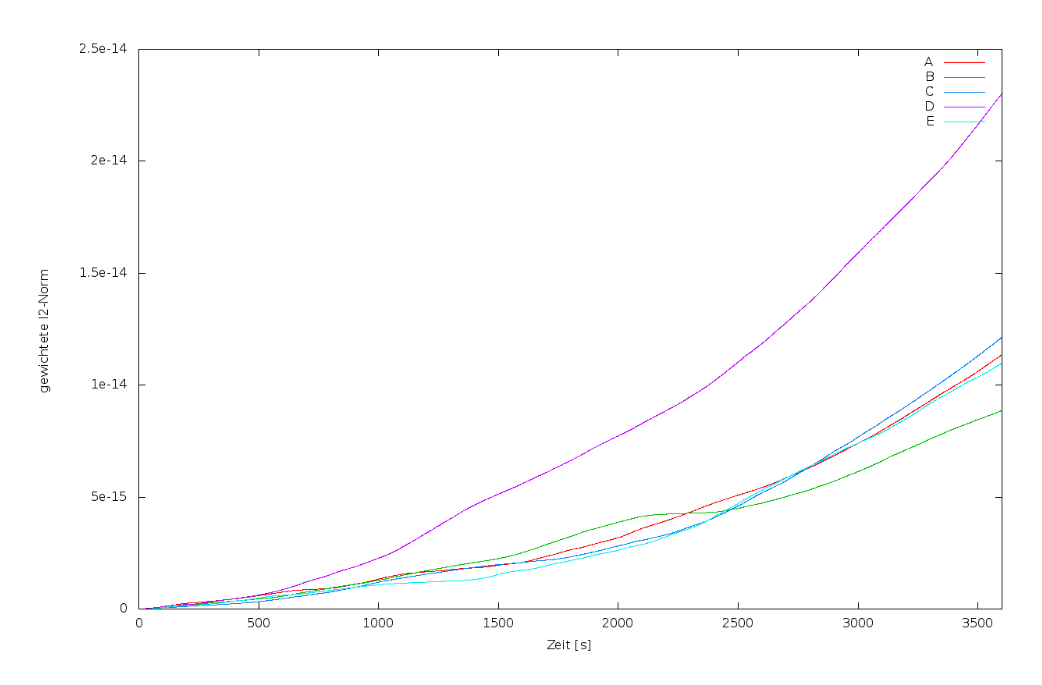


Abbildung 2.5: Verifikation verschiedener Parametersätze für den Symplektischen Integrator 8.Ordnung über einen Zeitraum von einer Stunde

Um zu noch höheren Ordnungen vorzudringen hat [Yoshida90] folgenden Hinweis gegeben:
In order to obtain 10th order integrators in this way, the next order of the BCH formulae becomes necessary, and the order of the simultaneous algebraic equations becomes much higher.

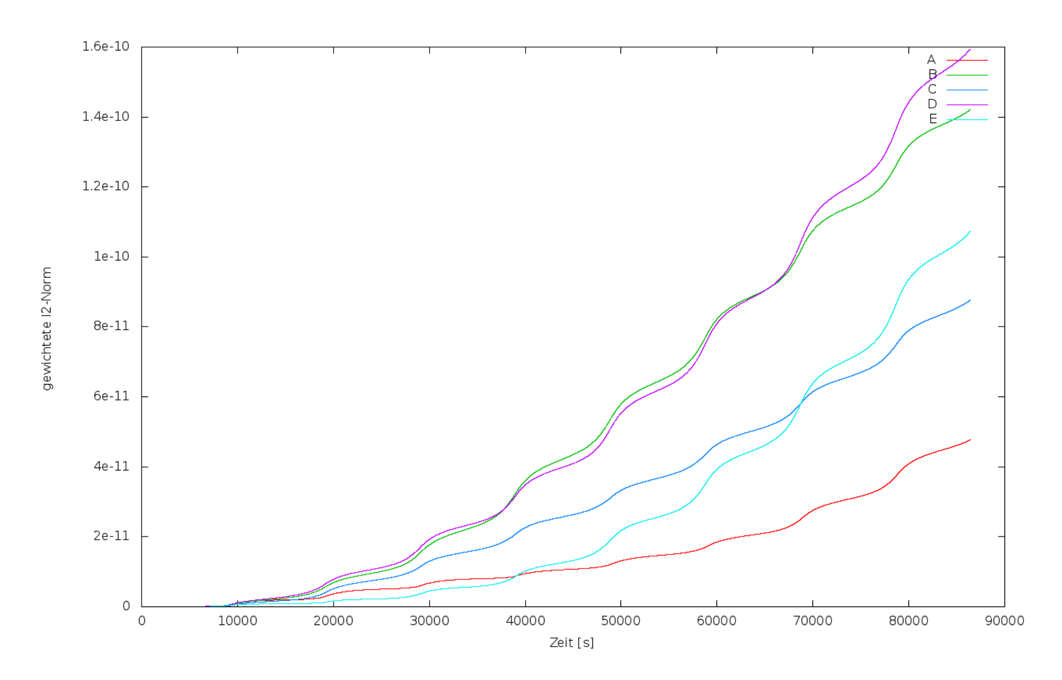


Abbildung 2.6: Verifikation verschiedener Parametersätze für den symplektischen Integrator 8.Ordnung über einen Zeitraum von einem Tag

2.3.5.1 Das Leap-Frog Integrationsverfahren [LF]

Dieses Einschrittverfahren[12] ist speziell zu Lösung physikalischer Bewegungsgleichungen konzipiert und stellt eine Verbesserung des Euler-Verfahrens dar. Hierbei wird abwechselnd der Ort $y(t_i)$ und die Geschwindigkeit $\frac{dy(i+\frac{h}{2})}{dt_{i+\frac{h}{2}}} = \dot{y}(i+\frac{h}{2})$ zu unterschiedlichen Zeitpunkten t_i und $t_{i+\frac{1}{2}}$ berechnet, wobei die Schrittweite mit h bezeichnet wird. Bei diesem Verfahren werden auch die Anfangswerte zu unterschiedlichen Zeitpunkten benötigt, d.h. beispielsweise der Ort $y(1)$ und die Geschwindigkeit $\dot{y}(1.5)$ mit $h = 1$ als Schrittweite. Das Prinzip ist in Abb.2.7 illustriert, woraus auch die Namenswahl dieses Verfahrens ersichtlich wird. Dieses Verfahren ist von zweiter Ordnung und benötigt pro Integrationsschritt eine Auswertung der rechten Seite der Kraftfunktion. Die Berechnung eines Integrationsschritts, beginnend vom Zeitpunkt t_0 mit einer Schrittweite $h = 1$, kann formal wie folgt beschrieben werden:

$$\begin{aligned} Y(t_1) &= Y(t_0) + h\dot{Y}(t_{\frac{1}{2}}) \\ \dot{Y}(t_{\frac{3}{2}}) &= \dot{Y}(t_{\frac{1}{2}}) + hf(Y(t_0), \dot{Y}(t_{\frac{1}{2}})) \end{aligned} \tag{2.3.53}$$

Dies kann weiter verallgemeinert werden durch

$$\begin{aligned} Y(t_{i+1}) &= Y(t_i) + h\dot{Y}(t_{i+\frac{1}{2}}) \\ \dot{Y}(t_{i+\frac{3}{2}}) &= \dot{Y}(t_{i+\frac{1}{2}}) + hf(Y(t_i), \dot{Y}(t_{i+\frac{1}{2}})) \end{aligned} \tag{2.3.54}$$

wobei $i \in N$ und $i \geq 0$ gilt. Ein verwandtes Verfahren, das hier noch erwähnt werden soll, ist das Verlet-Verfahren. Hierbei handelt es sich um ein Verfahren 4.Ordnung,

[12] siehe mehr dazu unter [Hut95] und [Mikkola01]

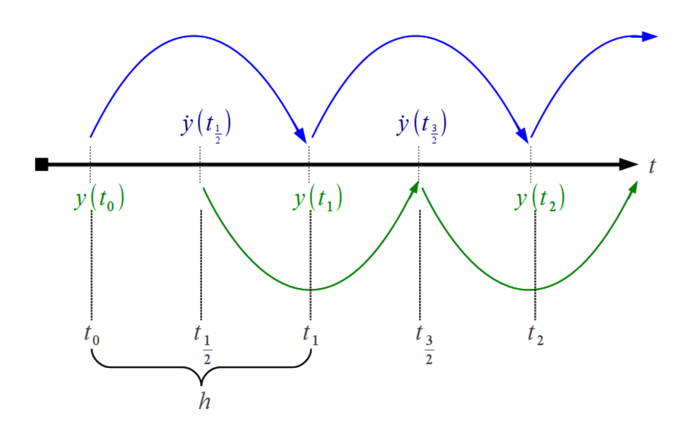


Abbildung 2.7: Prinzip der Leapfrog Integration

welches ebenfalls zur Gruppe der symplektischen Integrationsverfahren zählt. Weitere Informationen zu diesem Verfahren sind in [Verlet67] und [Hairer03] zu finden.

Beispiel 2.6 für „Asynchrone Anfangs-u. Ausgabewerte beim LF-Integrator“
Wie bereits erwähnt, erfordert die Integration mit dem LF-Integrator zeitlich um $\frac{h}{2}$ versetzte Anfangswerte für Ort und Geschwindigkeit. In Abbildung 2.8 (Seite 24) wurde die y-Komponente eines Kepler-Orbits über einen kurzen Zeitraum von 10 Sekunden integriert und dieser in Ort $y(t)$ und Geschwindigkeit $v(t)$ über der Zeit aufgetragen. Dieses Verfahren liefert, analog zu den Anfangswerten, auch Stützstellen zu unterschiedlichen Zeitpunkten. Damit nimmt dieses Verfahren, verglichen mit allen anderen hier erwähnten Verfahren, eine Sonderstellung ein.

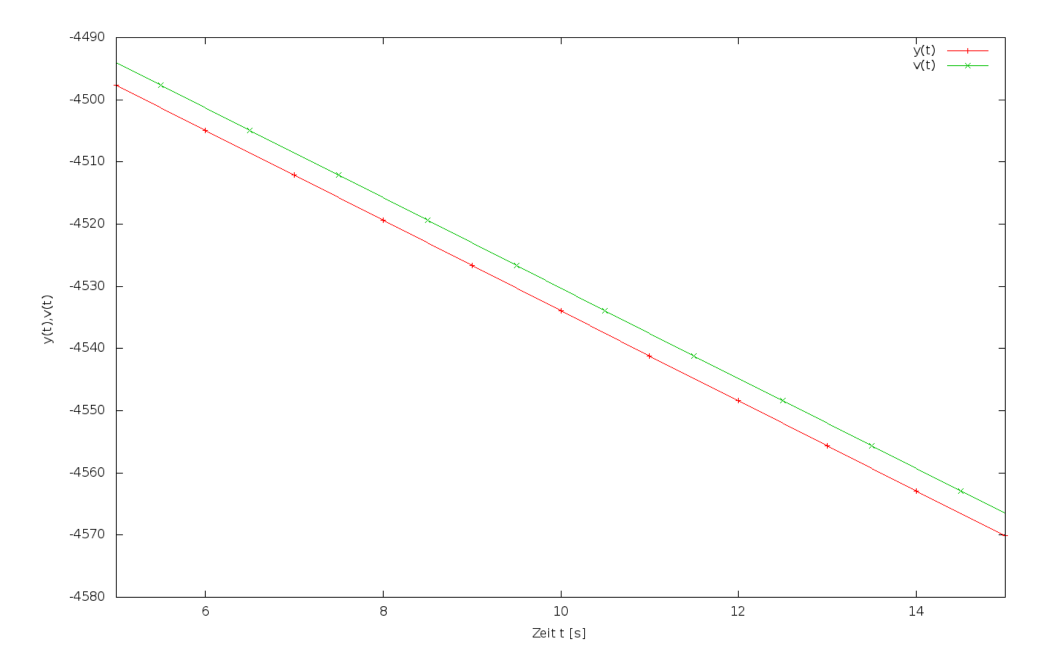


Abbildung 2.8: Integration mit dem Leapfrog Integrator

2.3.5.2 Das Verfahren von Gauss - Jackson [GJ4]

[GJ4] Gauss - Jackson 4. Ordnung

Das Gauss-Jackson-Verfahren ist ein Mehrschritt-Prädiktor-Verfahren mit fester Schrittweite h, welches zur Lösung von Differentialgleichungen 2.Ordnung konzipiert wurde. Dieses Verfahren ist wie das Burlisch-Stoer und das Shampine-Gordon-Verfahren nicht selbststartend und benötigt zum Anlauf der Integration eine bestimmte Anzahl von Stützstellen zurückliegender Zeitpunkte $(..., t_0 - 2h, t_0 - h, t_0, t_0 + h, t_0 + 2h, ...)$. Diese Schritte können für $n \in [-2, 2]$ und $n \in N$ kurz als $t_n = t_0 + nh$ ausgedrückt und mit $t_{-2}, t_{-1}, t_0, t_1, t_2$ geschrieben werden. Damit nur ein Anfangswertpaar aus Ort und Geschwindigkeit zu Beginn benötigt wird, werden diese mit Hilfe des RK4-Integrators in einer Initialisierungsroutine berechnet. Aus diesen Startwerten wird dann eine Tabelle aus Rückwärtsdifferenzen aufgebaut, die bei jedem Integrationsschritt aktualisiert wird. Weitere Informationen zur Herleitung, als auch Hinweise zur Implementierung des GJ4-Integrators, sind in [Berry04] und in [Montenbruck05] (Kapitel 4.2.7) zu finden.

2.3.5.3 Das Extrapolationsverfahren von Burlisch und Stoer [BS]

[BS] Burlisch und Stoer

Das BS-Verfahren benutzt im Prädiktorschritt der Integration das MMID-Verfahren[13], kombiniert mit der Richardson Extrapolationsmethode. Dabei wird eine vorgegebene Schrittweite h während der Integration sukzessiv in kleinere Schrittweiten $\tilde{h}$ unterteilt. Es werden folgende Unterteilungen verwendet

$$n = 2, 4, 6, 8, 12, 16, 24, 32, ..., (n_j = 2nj - 2) \tag{2.3.55}$$

Bei jeder Unterteilung wird aus den berechneten Funktionswerten ein Polynom approximiert. Im Anschluss wird der verbleibende Fehler abgeschätzt. Ist dieser größer als eine vorgegebene Fehlerschranke, so wird das Intervall erneut mit dem nächsthöheren n_j unterteilt und daraus ein weiteres Polynom approximiert. Um eine beliebige Verkleinerung der Schrittweite $\tilde{h}$ zu vermeiden, wurde eine Obergrenze festgelegt. Dieses Verfahren verfügt außerdem über adaptive SWS. Dabei wird die Schrittweite h entsprechend der Fehlerabschätzung adaptiv angepasst[14]. Die Grundidee des BS-Verfahrens basiert auf der Kombination von drei verschiedenen Verfahren.

[SWS] Schrittweitensteuerung

1. Die Richardson Extrapolation[15]
   Mit dieser Methode wird versucht, mit Hilfe mehrerer numerisch berechneter Funktionswerte einen genaueren Funktionswert zu einem späteren Zeitpunkt zu extrapolieren. Die jeweils benötigten Funktionswerte werden anhand der Modified Midpoint Methode (mehr dazu im Abschnitt 2.3.4.6) ausgewählt und berechnet. Diese werden schließlich als Stützstellen für einen numerischen Ausgleich verwendet. Die Methode der Richardson Extrapolation und deren Funktionsweise ist ein integraler Bestandteil des BS-Integrators. Darum wird im folgenden Unterpunkt (Seite 26) die Funktionsweise erläutert und dies mit einem Beispiel untermauert.

[13] siehe dazu [Press92]
[14] siehe dazu [Press92] und [Press07]
[15] siehe mehr dazu unter [Sidi10]

2. Wahl der numerischen Ausgleichsmethode
   Die bereits erwähnte Ausgleichung der numerisch bestimmten Funktionswerte kann auf verschiedene Art und Weise erfolgen. Es stehen prinzipiell zwei verschiedene Varianten zur Auswahl: polynom - oder rationaler Ausgleich. Hierfür wird in der Literatur[16] ersterer bevorzugt, die zweite aber dennoch angegeben. Aus diesem Grund wird in einem weiteren Beispiel deren Eignung untersucht.

3. Der Einsatz der Modified Midpoint Methode mit deren Eigenschaften
   Bei dieser Methode wird die Fehlerfunktion als eine Potenzreihe betrachtet, die nur gerade Potenzen besitzt. Indem ein höherer Term der Fehlerfunktion eliminiert wird, gewinnt man zwei Ordnungen an Genauigkeit. An dieser Stelle wird auf den Abschnitt 2.3.4.6 verwiesen. Dort wird dieses Verfahren detailliert erläutert.

Die Richardson Extrapolation

Diese Methode kann verwendet werden, um numerisch bestimmte Resultate zu verbessern. Durch Anwendung dieses Verfahrens kann eine zusätzliche Ordnung $\mathcal{O}(p+1)$ hinzugewonnen werden. Ausgegangen wird davon, dass eine numerisch angenäherte Lösung $Y(h)$ in Abhängigkeit einer bestimmten Schrittweite h berechnet wurde. Dies kann wie folgt ausgedrückt werden:

$$y = Y(h) + \mathcal{O}(h^p) \tag{2.3.56}$$

Hierbei ist $\mathcal{O}(h^p)$ die Fehlerordnung in Abhängigkeit der verwendeten Schrittweite h. In vielen Fällen kann dieser Zusammenhang um ein von h unabhängigen Faktor u erweitert werden, sodass sich folgender erweiterter Ausdruck ergibt:

$$y = Y(h) + uh^p + \mathcal{O}(h^p) \tag{2.3.57}$$

Indem zwei approximierte Lösungen mit unterschiedlichen Schrittweiten h_1 und h_2 bestimmt werden, wird versucht, den unabhängigen Faktoren u zu eliminieren und dabei die Fehlerordnung zu gewinnen. Sind die zwei unterschiedlichen Lösungen berechnet, so ergibt sich folgender Zusammenhang:

$$\begin{aligned} y &= Y(h_1) + uh_1^p + \mathcal{O}(h_1^p) \\ y &= Y(h_2) + uh_2^p + \mathcal{O}(h_2^p) \end{aligned} \tag{2.3.58}$$

Die Gleichungen können vereinfacht werden, indem das h_1^p -fache der zweiten Gleichung vom h_2^p -fachen der ersten Gleichung abgezogen wird. Dies ergibt:

$$y(h_2^p - h_1^p) = h_2^p Y(h_1) - h_1^p Y(h_2) + \mathcal{O}(h^{2p+1}) \tag{2.3.59}$$

Dies ergibt folgenden Ausdruck:

$$y_r = \frac{h_2^p Y(h_1) - h_1^p Y(h_2)}{(h_2^p - h_1^p)} + \mathcal{O}(h^{p+1}) \tag{2.3.60}$$

[16] siehe [Press92] auf Seite 731 (unten)

wobei das Ergebnis der Richardson Extrapolation als y_r bezeichnet wird.
Die Schrittweiten h_1 und h_2 können prinzipiell frei gewählt werden. Setzt man die Schrittweiten in Relation und werden beispielsweise für $h_1 = h$ und $h_2 = \frac{h}{2}$ verwendet, so kann dieser Ausdruck dargestellt werden als:

$$y_r = \frac{(\frac{h}{2})^p Y(h) - h^p Y(\frac{h}{2})}{(\frac{h}{2})^p - h^p} + \mathcal{O}(h^{p+1}) \quad (2.3.61)$$

Nun soll die Arbeitsweise der Richardson-Extrapolation exemplarisch dargestellt werden:

Beispiel 2.7 für „Extrapolation von e^x mit Hilfe der Richardson Extrapolationsmethode“
In diesem Beispiel soll die Exponentialfunktion e^x mit Hilfe der Richardson Extrapolationsmethode annähernd bestimmt werden. Die Exponentialfunktion kann mit folgender Funktion $Y(h)$ näherungsweise beschrieben werden:

$$Y(h) = (1 + xh)^{\frac{1}{h}} \quad (2.3.62)$$

Somit können zwei Lösungen, abhängig von der Schrittweite h berechnet werden. Wird $h = 0.1$ und $x = 1$ gewählt, so ergeben sich folgende Zwischenergebnisse:

$$\begin{aligned} Y(h_1 = 0.1) &= 2.5937424601 \\ Y(h_2 = 0.05) &= 2.6532977051 \end{aligned} \quad (2.3.63)$$

Durch Einsetzen in die Formel 2.3.61 mit den Parameter $p = 1$ ergibt folgendes Ergebnis für y_r:

$$\begin{aligned} y_r &= 2.712852950 \\ e^1 &= 2.718281828 \end{aligned} \quad (2.3.64)$$

Hierbei wurden die nicht korrekten Dezimalstellen rot eingefärbt. In der zweiten Zeile befindet sich das korrekte Ergebnis für die Exponentialfunktion an der Stelle $x = 1$.

Um herauszufinden welche Ausgleichsmethode für einen konkreten Fall geeignet ist und inwieweit das berechnete Ergebnis vom erwarteten abweicht, wird im Folgenden ein weiteres Experiment durchgeführt. Als zusätzliche Motivation für nachfolgende Überprüfung folgt ein Zitat aus *Numerical Recipies*[17] zur Wahl der Ausgleichsmethode beim BS-Integrator:

Current wisdom favors polynomial extrapolation over rational function extrapolation in the Bulirsch-Stoer method. However, our feeling is that this view is guided more by the kinds of problems used for tests than by one method being actually better.

Beispiel 2.8 für „Untersuchung zur Wahl der Ausgleichsmethode“
Hierfür soll am Beispiel des ungestörten harmonischen Oszillators[18] überprüft werden, welche Ausgleichsmethode zur Lösung dieser konkreten Aufgabe die bessere ist. Der ungestörte harmonische Oszillator wurde ausgewählt, da hierfür sowohl eine gewöhnliche Differentialgleichung als auch eine analytische Lösung bereitsteht. Somit kann das Ergebnis des BS-Integrators in Abhängigkeit der verwendeten Ausgleichsmethode anhand der analytisch berechneten Lösung verifiziert werden. Die Differentialgleichung des harmonischen Oszillators

[17] entnommen aus [Press92] Seite 731, unten
[18] siehe dazu mehr unter [Taylor05] ab Seite 163

ist wie folgt definiert:

$$\frac{d^2y(t)}{dt^2} = -\frac{k}{m}y(t) \tag{2.3.65}$$

Mit m als Masse, k als Federkonstante und t als Zeitpunkt. Dies kann wiederum umgeformt werden als

$$\frac{d^2y(t)}{dt^2} = -\omega^2 y(t), \qquad \omega = \sqrt{\frac{k}{m}} \tag{2.3.66}$$

Die entsprechende analytische Lösung $y(t)$ besitzt folgende Form

$$y(t) == A\cos(\omega t + \phi), \phi = 0 \tag{2.3.67}$$

Wobei A die Amplitude der analytischen Funktion ist. Wird nun die Differentialgleichung des harmonischen Oszillators mit dem BS-Integrator mit verschiedenen Ausgleichsmethoden gelöst, so ergibt sich folgender Unterschied (siehe Abb. 2.9). Hierbei wurde die y-Achse logarithmisch skaliert und der kumulative absolute Fehler beider Methoden über der Zeit aufgetragen. In diesem Fall unterscheiden sich die beiden Lösungen schon nach 100 Sekunden um zwei Dezimalstellen. Aus diesem Grund ist stets zu überprüfen, welche Art der Ausgleichsmethode für den jeweiligen Fall verwendet werden soll. Für diesen Fall zeigt die rationale Methode das bessere Fehlerverhalten.

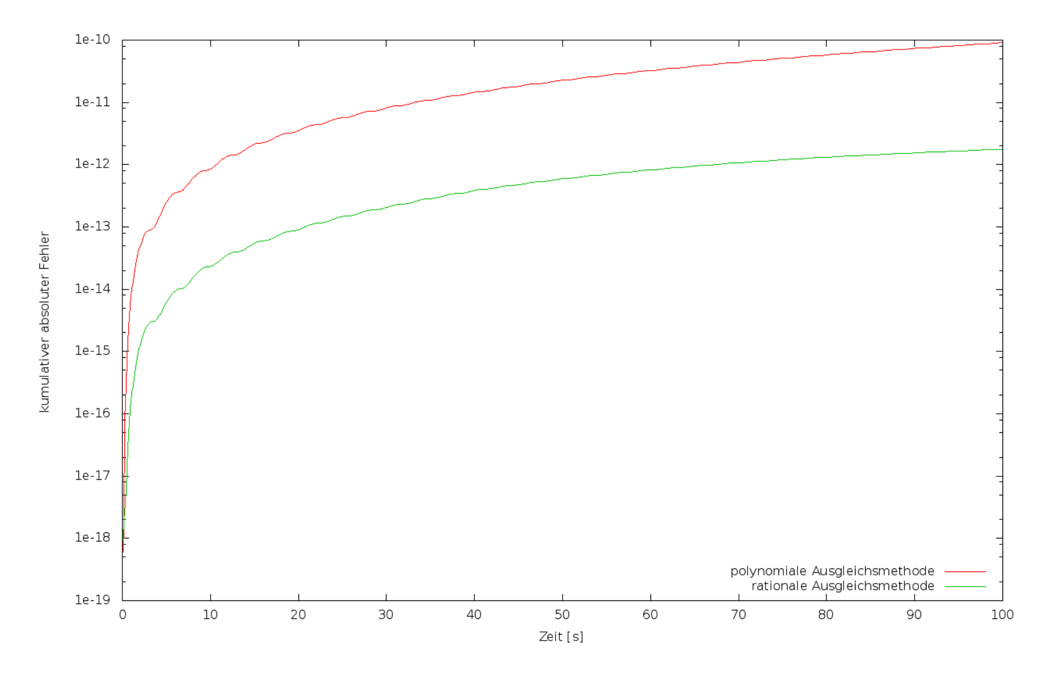


Abbildung 2.9: Gegenüberstellung der Ergebnisse bei Verwendung unterschiedlicher Berechnungsverfahren beim Burlisch-Stoer-Integrator [BS]

2.3.5.4 Das Mehrschrittverfahren nach Shampine und Gordon [SG]

[SG] Shampine Gordon

Das SG-Verfahren basiert auf dem Verfahren von Adams Bashforth und Adams Moulton. Es benötigt zwei Funktionsaufrufe, um einen Integrationsschritt durchzuführen.

Jeweils einen Prädiktor und einen Korrektor-Schritt. Es ist darüber hinaus mit adaptiver Schrittweitensteuerung ausgestattet. Dazu besitzt das SG-Verfahren eine Ordnungssteuerung, die beim Auftreten einer Unstetigkeit mit dem Anlaufverfahren neu beginnt[19].
Die implementierten Verfahren und deren Einordnung nach den besprochenen Kriterien sind in Abbildung 2.15 auf Seite 44 dargestellt.

2.3.6 Quadratur

Die Quadratur[20] stellt prinzipiell eine weitere Möglichkeit zur Lösung gewöhnlicher Differentialgleichungen dar. Sie wird oft zur Integration von Funktionen verwendet, für die keine analytische Stammfunktion vorhanden sind. Diese Gruppe von Lösungsverfahren wird hier nur am Rande betrachtet, es wird aber dennoch das Prinzip vorgestellt. Hierbei soll das folgende unbestimmte Integral $I(f)$ gelöst werden:

$$I(f) = \int_a^b f(x)dx = F(b) - F(a) \tag{2.3.68}$$

mit $F(a)$ und $F(b)$ als Stammfunktion. In vielen Fällen lassen sich keine Stammfunktionen in geschlossener Form angeben und man ist auf Näherungsverfahren angewiesen. Bei der Gauß-Quadratur[21] wird die Funktion zunächst an bestimmten Stellen (x_i) ausgewertet. Mit Hilfe dieser Werte und den Gewichten ω_i wird auf Basis der Quadraturregel[22]:

$$I(f) = \int_a^b f(x)dx \approx \underbrace{\sum_{i=0}^{n} \omega_i f(x_i)}_{\text{Quadraturformel}} = \underbrace{I_n(f)}_{\text{Näherungslösung}} \tag{2.3.69}$$

eine Lösung bestimmt. Hierfür wurden in der Vergangenheit eine Vielzahl verschiedener Methoden entwickelt. Beispiele dafür sind außerdem die Gauß-Lengendre und die Gauß-Tschebyscheff-Quadratur, als auch die Romberg-Integration. Weitere Informationen zu diesen und weiteren Varianten findet man in [Press07], [Bronstein08] aber auch in [Deuflhard08] (ab Seite 294).

2.4 Potenzreihenintegration

Die Potenzreihenintegration stellt ein alternatives Werkzeug zur Lösung gewöhnlicher Differentialgleichungen dar. Im Gegensatz zu den in Abschnitt 2.3 beschriebenen numerischen Integrationsverfahren wird bei der Potenzreihenintegration während eines

[19] siehe dazu [Montenbruck05]
[20] siehe mehr dazu unter [Bronstein08]
[21] Die entsprechenden Funktionen zur Gauß-Quadratur wurden im Rahmen dieser Arbeit implementiert und durch Kontrollrechnungen mit den Ergebnissen von Herrn Prof. Ilk (Universität Bonn) Fortran Implementierung verifiziert.
[22] entnommen aus [Huckle06], Seite 176

Integrationsschritts kein diskreter Punkt berechnet, sondern die Taylorreihenentwicklung der gesuchten Lösung bestimmt[23]. Die Entwicklung einer Taylorreihe hat gegenüber den klassischen Methoden den entscheidenden Vorteil, dass bei jedem Integrationsschritt eine beliebige Anzahl von Reihenkoeffizienten berechnet werden kann. Es ist somit ein wertvolles Werkzeug zur Verifikation der Standardverfahren, da hier nur durch einfaches Erhöhen des Entwicklungsgrads die Genauigkeit der Lösung gesteigert werden kann[24].

2.4.1 Rechnung mit Potenzreihen

Das Verfahren der Integration von Potenzreihen basiert auf der Verknüpfung von Reihen durch mathematische Operationen. Nachfolgend sollen die beiden Funktionen $a(t)$ und $b(t)$ als Reihenentwicklung um den Entwicklungspunkt t_0 definiert werden[25]:

$$\begin{aligned} a(t) &= \sum_{m=0}^{\infty} \langle a_m \rangle (t-t_0)^m, \qquad \langle a_m \rangle = \frac{1}{m!}\frac{d^m a}{dt^m}|_{t=t_0} \\ b(t) &= \sum_{m=0}^{\infty} \langle b_m \rangle (t-t_0)^m, \qquad \langle b_m \rangle = \frac{1}{m!}\frac{d^m b}{dt^m}|_{t=t_0} \end{aligned} \tag{2.4.70}$$

Werden nun die beiden Potenzreihen $a(t)$ und $b(t)$ mit einer arithmetischen Operation verknüpft, so entsteht eine neue Potenzreihe $c(t)$. Dies kann wie folgt ausgedrückt werden:

$$c(t) = a(t) \circ b(t), \qquad \circ \in (+, -, *, /, \sqrt{}, exp, ...) \tag{2.4.71}$$

wobei die Koeffizienten $\langle c_m \rangle$ durch nachfolgend beschriebene, meist rekursive Bezie-

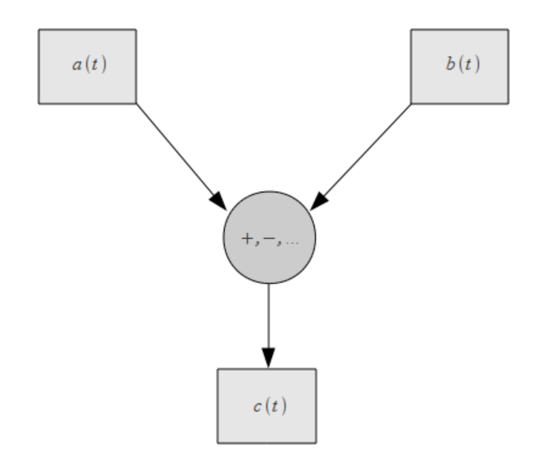


Abbildung 2.10: Verknüpfung von Potenzreihen

hungen, angegeben werden können. Zur besseren Vorstellung sei die Verknüpfungsvorschrift in Abb. 2.10 illustriert. Die rekursiven Beziehungen zur Verknüpfung von Potenzreihenkoeffizienten sind ebenfalls in [Wanner68] (ab Seite 10) und in [Schneider92] (auf Seite 396 und 397) aufgelistet.

[23] siehe dazu [Montenbruck91]

[24] Dies gilt nur innerhalb des Konvergenzradius der jeweiligen Reihe.

[25] Hierbei wurde die Notation aus [Montenbruck91] (ab Seite 7) verwendet.

2.4.2 Rekursionsformeln zur Verknüpfung von Potenzreihen

Alle Koeffizienten $\langle c \rangle_m$ einer Taylorreihe m-ter Ordnung können wie folgt geschrieben werden:

$$\langle c \rangle_m = \langle c_0 \rangle ... \langle c_m \rangle \tag{2.4.72}$$

Dabei ist beispielsweise $\langle c_0 \rangle$ der erste Koeffizient und $\langle c_m \rangle$ der letzte Koeffizient der Reihe. In Abbildung 2.11 sind die Potenzreihenkoeffizienten illustriert.

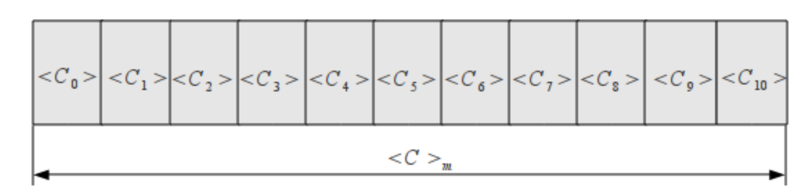


Abbildung 2.11: Illustration der Potenzreihenkoeffizienten

2.4.2.1 Addition und Subtraktion

Potenzreihenkoeffizienten können analog zu Spaltenvektoren gliedweise addiert und subtrahiert werden und es gilt folgende Verknüpfungsvorschrift:

$$\langle c_m \rangle = \langle a_m \rangle \pm \langle b_m \rangle \tag{2.4.73}$$

2.4.2.2 Multiplikation

Die Multiplikation, auch als Cauchy-Faltung bekannt, ist wie folgt definiert:

$$\langle c_m \rangle = \sum_{i=0}^{m} \langle a_i \rangle \langle b_{m-i} \rangle \tag{2.4.74}$$

2.4.2.3 Division

Die Division $c(t) = \frac{a(t)}{b(t)}$ kann aus der Multiplikation hergeleitet werden und ergibt:

$$\langle c_m \rangle = \frac{1}{\langle b_0 \rangle} \left(\langle a_m \rangle - \sum_{i=0}^{m-1} \langle c_i \rangle \langle b_{m-i} \rangle \right) \tag{2.4.75}$$

Hierbei gibt es einige Unterschiede zu den Grundrechenarten:

- Bei der Division zweier Potenzreihen besteht eine Abhängigkeit der $\langle c_m \rangle$-Terme. Diese können nur in aufsteigender Reihenfolge (Index $0 ... m-1$) berechnet werden. Die anderen Grundrechenarten besitzen diese Abhängigkeit nicht und können prinzipiell auch parallel berechnet werden.

- Der Rechenaufwand zur Berechnung eines Potenzreihenkoeffizienten im Falle der Grundrechenarten Addition und Subtraktion unterscheidet sich nicht von der herkömmlichen Rechnungen. Im Gegensatz dazu wächst bei der Multiplikation

bzw. Division der Rechenaufwand etwa proportional zur Ordnung[26]. Der Mehraufwand bei der Verknüpfung zweier Potenzreihen der Ordnung m steigt für Additions - u. Subtraktionsoperationen linear. Bei Multiplikation und Division wächst der Aufwand quadratisch mit der Ordnung der Reihenentwicklung.

2.4.2.4 Potenzbildung

Die Potenzbildung kann mit Hilfe der Formel zur Multiplikation (Cauchy-Faltung) hergeleitet werden:

$$c(t) = a(t)^d, \qquad d \in N \wedge d \geq 0 \tag{2.4.76}$$

Durch sukzessives hintereinander Anwenden der Vorschrift zur Multiplikation ergibt sich folgender Ausdruck:

$$\begin{aligned} \langle c_m \rangle &= \frac{1}{m} \frac{1}{\langle a_0 \rangle} \left(\sum_{i=0}^{m-1} (d(m-i) - i) \langle c_i \rangle \langle a_{m-i} \rangle \right), \quad m \in N_+ \wedge m, \langle a_0 \rangle \neq 0 \\ \langle c_0 \rangle &= \langle a_0 \rangle^d \end{aligned} \tag{2.4.77}$$

2.4.2.5 Berechnung der Quadratwurzel

Die Quadratwurzel einer Potenzreihe $c(t) = \sqrt{a(t)}$ kann durch folgende Vorschrift bestimmt werden:

$$\begin{aligned} \langle c_m \rangle &= \frac{1}{2\langle a_0 \rangle} \left(\langle a_m \rangle - \sum_{i=1}^{m-1} \langle a_i \rangle \langle m-i \rangle \right), \quad \langle a_0 \rangle \neq 0 \\ \langle c_0 \rangle &= \sqrt{\langle a_0 \rangle} \end{aligned} \tag{2.4.78}$$

2.4.2.6 Differentiation

Die erste Ableitung der Potenzreihe $a'(t) = \frac{da(t)}{dt}$ kann nach [Montenbruck91] wie folgt hergeleitet werden:

$$\begin{aligned} a'(t) &= \sum_{m=0}^{\infty} \langle a'_m \rangle (t - t_0)^m \\ &= \frac{d}{dt} \sum_{m=0}^{\infty} \langle a_m \rangle (t - t_0)^m \\ &= \sum_{m=0}^{\infty} (m+1) \langle a_{m+1} \rangle (t - t_0)^m \end{aligned} \tag{2.4.79}$$

Danach gilt folgende Beziehung für jeden Potenzreihenkoeffizienten:

$$\langle a'_i \rangle = (i+1) \langle a_{i+1} \rangle \tag{2.4.80}$$

[26] definiert in [Montenbruck91]

Diese Eigenschaft der Potenzreihen erlaubt ein einfaches Differenzieren einer gesamten Potenzreihe, ohne großen Rechenaufwand. Dabei ist dennoch Vorsicht geboten, da in der Praxis die Ordnung m endlich ist und sich durch jede Differentiation die Ordnung der resultierenden Reihe um eine Ordnung vermindert. Dieser *Schwund* an Potenzreihenkoeffizienten muss bei der Implementierung berücksichtigt werden.

2.4.2.7 Integration

Analog zur Differentiation kann die Integration eines Potenzreihenkoeffizienten wie folgt definiert werden:

$$\begin{aligned} \int \langle a_i \rangle &= \frac{\langle a_i \rangle}{i+1}, \quad i \in N_+ \\ \int \langle a_0 \rangle &= \langle -a_0 \rangle \end{aligned} \tag{2.4.81}$$

2.4.2.8 Trigonometrische Funktionen

Zur Berechnung des Sinus bzw. Cosinus einer Potenzreihe $b(t) = sin(a(t))$ bzw. $c(t) = cos(a(t))$ kann dies durch folgendes Formelpaar ausgedrückt werden[27]:

$$\begin{aligned} \langle sin(a(t))_m \rangle &= \frac{1}{m} \sum_{i=0}^{m-1} (m-i) \langle a_{m-i} \rangle \langle cos(a(t)_i \rangle \\ \langle cos(a(t))_m \rangle &= -\frac{1}{m} \sum_{i=0}^{m-1} (m-i) \langle a_{m-i} \rangle \langle sin(a(t)_i \rangle \end{aligned} \tag{2.4.82}$$

Hierfür werden folgende Startbedingungen benötigt:

$$\begin{aligned} \langle sin(a(t)_0) \rangle &= sin(\langle a(t)_0 \rangle) \\ \langle cos(a(t)_0) \rangle &= cos(\langle a(t)_0 \rangle) \end{aligned} \tag{2.4.83}$$

Ferner sind in [Wanner68] (ab Seite 31) für die trigonometrischen Funktionen tan, $atan$, $asin$ und $acos$ die rekursiven Beziehungen angegeben.

2.4.2.9 Die Exponentialfunktion

Die Exponentialfunktion $c(t) = exp(a(t))$ kann wie folgt auf Potenzreihenkoeffizienten angewandt werden:

$$\begin{aligned} \langle c_m \rangle &= \frac{1}{m} \sum_{i=0}^{m-1} (m-i) \langle c_i \rangle \langle a_{m-1} \rangle \\ \langle c_0 \rangle &= exp(\langle a_0 \rangle) \end{aligned} \tag{2.4.84}$$

[27] Siehe mehr dazu in [Schneider93]

2.4.2.10 Logarithmus

Der Logarithmus einer Potenzreihe $c(t) = log(a(t))$ kann mit folgendem formelmäßigen Zusammenhang ausgedrückt werden:

$$\langle c_m \rangle = \frac{1}{\langle a_0 \rangle} \left(\langle a_m \rangle \sum_{i=1}^{m-1} (m-i) \langle a_i \rangle \langle c_{m-1} \rangle \right), \quad \langle a_0 \rangle \neq 0$$
$$\langle c_0 \rangle = log(\langle a_0 \rangle) \tag{2.4.85}$$

Bei der Berechnung ist wieder die Abhängigkeit der einzelnen Potenzreihenkoeffizienten der Berechnungsvorschrift besonders hervorzuheben. Hier ist analog zur Division und der Exponentialfunktion keine parallele Berechnung der Koeffizienten möglich, da die jeweils vorangegangenen Koeffizienten zur Berechnung des Koeffizienten der nächsthöheren Ordnung benötigt werden.
Die nachfolgend angegebenen rekursiven Beziehungen stellen Erweiterungen zu den bereits bekannten Grundfunktionen dar. Sie wurden zur Steigerung der Programmeffizienz bei der programmiertechnischen Umsetzung von Potenzreihenverknüpfungen ausgearbeitet.

2.4.2.11 Fused multiplication and addition (fma)

Diese Befehlserweiterung wird von einigen Prozessorherstellern angeboten, um drei Gleitkommazahlen $(a + b) * c$ mit nur einem Befehlsaufruf miteinander verknüpfen zu können. Dabei wird im Falle von Gleitkommazahlen i.d.R. nur ein Prozessortaktzyklus für die Berechnung benötigt, wohingegen ohne den Aufruf der speziellen fma-Funktion dafür nicht garantiert wird. Diese Namensbezeichnung wurde aufgegriffen und auf die Verknüpfung von Potenzreihen angewendet, um sowohl unnötige Funktionsaufrufe als auch unnötiges Kopieren von Potenzreihenkoeffizienten einzusparen. Somit kann die fma-Verknüpfung von $d(t) = a(t) * b(t) + c(t)$ wie folgt beschrieben werden:

$$\langle d_m \rangle = \sum_{i=0}^{m} (\langle a_i \rangle \langle b_{m-i} \rangle) + \langle c_i \rangle \tag{2.4.86}$$

Hier wird analog zu den anderen Verknüpfungsvorschriften vorausgesetzt, dass die Potenzreihen dieselbe Ordnung m besitzen. Analog zur fma-Verknüpfung, kann eine fms-Verknüpfung (Subtraktion) definiert werden durch :

fms: fused multiplication and subtraction

$$\langle d_m \rangle = \sum_{i=0}^{m} (\langle a_i \rangle \langle b_{m-i} \rangle) - \langle c_i \rangle \tag{2.4.87}$$

2.4.2.12 Häufig verwendete Potenzen $a(t)^2, a(t)^3$ und $a(t)^4$

Für häufig verwendete konstante Potenzen kann eine vereinfachte Vorschrift erstellt werden. Das Quadrieren $a(t)^2$ einer Potenzreihe ist eine Vereinfachung der Potenzbil-

dung:

$$\begin{aligned} \langle c_m \rangle &= \frac{1}{m} \frac{1}{\langle a_0 \rangle} \left(\sum_{i=0}^{m-1} (2(m-i)-i) \langle c_i \rangle \langle a_{m-i} \rangle \right), \quad m \in N_+ \wedge m, \langle a_0 \rangle \neq 0 \\ \langle c_0 \rangle &= \langle a_0 \rangle^2 \end{aligned} \tag{2.4.88}$$

Hinsichtlich einer effizienten Abarbeitungsgeschwindigkeit empfiehlt sich die Implementierung dieser Vorschriften für die jeweiligen Fälle. Dadurch können unnötige Funktionsaufrufe vermieden und folglich kostbare Rechenzeit eingespart werden.

2.4.2.13 Die l_2-Norm

Besonders bei der Lösung von Bewegungsproblemen, wie beispielsweise dem Keplerproblem, ist es üblich, den Ort x, y, z und die Geschwindigkeit $\dot{x}, \dot{y}, \dot{z}$ in dreidimensionaler Vektorschreibweise anzugeben. Hierbei wird häufig die L2-Norm $|r|$ benötigt, welche mathematisch wie folgt definiert ist:

$$|r| = \sqrt{x^2 + y^2 + z^2} \tag{2.4.89}$$

In Potenzreihendarstellung $r(t) = \sqrt{x^2(t) + y^2(t) + z^2(t)}$ kann folgende Vorschrift zur Berechnung der jeweiligen Koeffizienten angegeben werden. Diese muss in zwei Schritten erfolgen, da sowohl die Potenzbildung als auch die Quadratwurzel die bereits erwähnte nicht parallelisierbare Eigenschaft besitzen.
Schritt 1: $r(t) = (x^2(t) + y^2(t) + z^2(t))$

$$\begin{aligned} \langle r_m \rangle = & \frac{1}{m\langle x_0 \rangle} \left(\sum_{i=0}^{m-1} (2(m-i)-i) \langle r_i \rangle \langle x_{m-i} \rangle) \right) \\ & + \frac{1}{m\langle y_0 \rangle} \left(\sum_{i=0}^{m-1} (2(m-i)-i) \langle r_i \rangle \langle y_{m-i} \rangle) \right) \\ & + \frac{1}{m\langle z_0 \rangle} \left(\sum_{i=0}^{m-1} (2(m-i)-i) \langle r_i \rangle \langle z_{m-i} \rangle) \right) \\ \langle r_0 \rangle = & \langle x_0 \rangle^2 + \langle y_0 \rangle^2 + \langle z_0 \rangle^2 \end{aligned} \tag{2.4.90}$$

Schritt 2: $c(t) = \sqrt{r(t)}$

$$\begin{aligned} \langle c_m \rangle &= \frac{1}{2\langle r_0 \rangle} \left(\langle r_m \rangle - \sum_{i=1}^{m-1} \langle r_i \rangle \langle m-i \rangle \right), \quad \langle r_0 \rangle \neq 0 \\ \langle c_0 \rangle &= \sqrt{\langle r_0 \rangle} \end{aligned} \tag{2.4.91}$$

2.4.3 Potenzreihenintegration in der Praxis

Nachdem die nötigen Formeln zur Verknüpfung von Potenzreihen bekannt sind, kann das Integrationsverfahren von Potenzreihen beschrieben werden. Hierbei wird analog zu den anderen Integrationsverfahren eine gewöhnliche Differentialgleichung ge-

löst. Die formelmäßige Darstellung der Potenzreihe wird hierfür als Verknüpfungsvorschrift verwendet, wobei jede Verknüpfung einer der hier definierten Formeln entspricht. Somit kann für jeden Integrationsschritt eine Potenzreihe m-ter Ordnung entwickelt und im Anschluss zum Zeitpunkt t_1 ausgewertet werden. Um das Verfahren zu veranschaulichen wird beispielhaft anhand der ungedämpften Duffingschen Oszillatorgleichung die Entwicklung einer Potenzreihe der Ordnung $m = 3$ schrittweise durchgeführt:

Beispiel 2.9 für „Der ungedämpfte Duffing Oszillator“
Die Entwicklung der Potenzreihe erfolgt in vier Schritten. Die Abarbeitungsreihenfolge wird durch die Rechengesetze der Grundrechenarten vorgegeben und ist in Abb. 2.12 dargestellt.

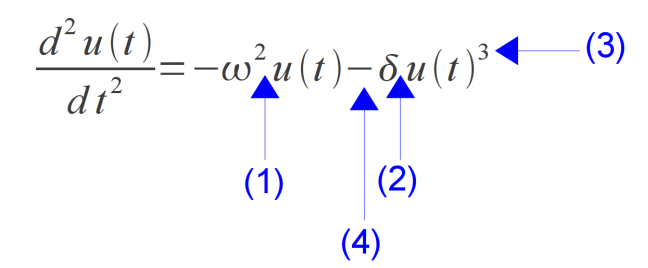


Abbildung 2.12: Formale Darstellung der Duffing-Oszillatorgleichung

Das Zwischenergebnis nach der (1). Verknüpfung r_1 und $\dot{r_1}$ hat folgende Gestalt:

$$\begin{aligned} r_1 &= u + \frac{du}{dt}h - \frac{1}{2}\omega^2 u h^2 \\ \dot{r_1} &= -uh\omega^2 + \frac{du}{dt} - \frac{1}{2}\frac{du}{dt}h^2\omega^2 \end{aligned} \tag{2.4.92}$$

Hierbei ist zu beachten, dass bei jeder Verknüpfung zwei Reihen entwickelt werden. Eine Reihe beschreibt den Ort (r_1) und die andere die Geschwindigkeit ($\dot{r_1}$). Die Zwischenergebnisse r_2 und $\dot{r_2}$ des zweiten Teilschritts (2) haben folgende Darstellung:

$$\begin{aligned} r_2 &= u + \frac{du}{dt}h - \frac{1}{2}u\delta h^2 \\ \dot{r_2} &= -u\delta h - \frac{1}{2}\frac{du}{dt}\delta h^2 + \frac{du}{dt} \end{aligned} \tag{2.4.93}$$

Beim nächsten Teilschritt (3) erfolgt eine Potenzbildung der Reihenkoeffizienten:

$$\begin{aligned} r_3 &= u + h\frac{du}{dt} - \frac{1}{2}h^2u^3\delta \\ \dot{r_3} &= -hu^3\delta - \frac{3}{2}h^2u^2\frac{du}{dt}\delta + \frac{du}{dt} \end{aligned} \tag{2.4.94}$$

Im letzten und (4).Schritt werden die Teilergebnisse r_2,r_3 und $\dot{r_2}$,$\dot{r_3}$ aufsummiert. Dies ergibt:

$$\begin{aligned} r_4 &= u - \frac{1}{2}uh^2\omega^2 + \frac{du}{dt}h - \frac{1}{2}u^3\delta h^2 \\ \dot{r_4} &= -\frac{3}{2}u^2\frac{du}{dt}\delta h^2 + \frac{du}{dt} - uh\omega^2 - u^3\delta h - \frac{1}{2}\frac{du}{dt}h^2\omega^2 \end{aligned} \tag{2.4.95}$$

2.4.4 Die Taylorreihenmethode

Die Taylorreihenmethode kann angewendet werden, falls genügend Ableitungsterme $\frac{dy(t)}{dt}$, $\frac{d^2y(t)}{dt^2}$, ..., $\frac{d^ny(t)}{dt^n}$ einer Funktion $y(t)$ vorhanden sind. Eine stetig differenzierbare Funktion ist dabei vorausgesetzt. Ist dies gegeben, können beispielsweise auf analytischem Weg höhere Ableitungsterme bestimmt werden und der Funktionswert zum Zeitpunkt $t + h$ mit Hilfe einer Taylorreihe bestimmt werden. Bei der Auswertung dieser muss jedoch auf den Konvergenzradius[28] r geachtet werden. Dieser beschränkt die Länge des Zeitschritts h und es gilt somit $h < r$. Ferner ist der Restgliedfehler zu beachten, d.h. wird die Reihenentwicklung bei einer bestimmten Ordnung n abgebrochen, dann liegt der Restgliedfehler in der Größenordnung des $n+1$-ten Reihenglieds. Das Auswerten der jeweiligen Reihe kann, ähnlich der Integration, in Schritten h unterschiedlicher Länge durchgeführt werden und stellt somit eine weitere Methode zur Lösung von Differentialgleichungen dar. Hierfür wurden für das Keplerproblem und das Hauptproblem der Himmelsmechanik die Taylorreihen-Koeffizienten formelmäßig erarbeitet. Diese sind im Anhang C und D aufgelistet.

Beispiel 2.10 für „Ein unangenehmes Beispiel“
Mit diesem Beispiel sollen die Grenzen einer Taylorreihenentwicklung aufgezeigt werden. Dazu wird ein populäres Beispiel aus der einschlägigen Fachliteratur herangezogen ([Collatz66], Seite 49):

$$\frac{d^2y(x)}{dx^2} = 10\frac{dy(x)}{dx} + 11y(x) \tag{2.4.96}$$

mit den Anfangswerten $y(x) = 1$, $\frac{dy(x)}{dx} = -1$ an der Stelle $x = 0$. Hieraus soll eine Taylorreihe entwickelt werden. Anschließend soll die Lösung an der Stelle $x = 1$ berechnet werden. Aus der angegebenen Differentialgleichung können nun höhere Ableitungsterme gebildet werden, um daraus eine Taylorreihe aufzubauen. Die Ableitungsterme können durch folgendes Bildungsgesetz ausgedrückt werden:

$$\begin{aligned}
\frac{d^3y(x)}{dx^3} &= 10\left(\frac{d^2}{d\,x^2}\,\mathrm{y}\,(x)\right) + 11\left(\frac{d}{d\,x}\,\mathrm{y}\,(x)\right)\\
\frac{d^4y(x)}{dx^4} &= 10\left(\frac{d^3}{d\,x^3}\,\mathrm{y}\,(x)\right) + 11\left(\frac{d^2}{d\,x^2}\,\mathrm{y}\,(x)\right)\\
&...\\
\frac{d^ny(x)}{dx^n} &= 10\left(\frac{d^{n-1}}{d\,x^{n-1}}\,\mathrm{y}\,(x)\right) + 11\left(\frac{d^{n-2}}{d\,x^{n-2}}\,\mathrm{y}\,(x)\right)
\end{aligned}$$

Wobei n der Platzhalter für den jeweiligen Grad der Ableitung ist. Daraus kann durch Einsetzen der Ableitungsterme folgende Taylorreihe n-ter Ordnung aufgebaut werden:

$$y(x) + \frac{dy(x)}{dx}\frac{x}{1!} + \frac{d^2y(x)}{dx^2}\frac{x^2}{2!} + ... + \frac{d^ny(x)}{dx^n}\frac{x^n}{n!}$$

Einsetzen der Ableitungsterme ergibt

$$y(x) + \frac{dy(x)}{dx}\frac{x}{1!} + 10\left(\frac{d^2}{d\,x^2}\,\mathrm{y}\,(x)\right) + 11\left(\frac{d}{d\,x}\,\mathrm{y}\,(x)\right)\frac{x^2}{2!} + ... + 10\left(\frac{d^{n-1}}{d\,x^{n-1}}\,\mathrm{y}\,(x)\right) + 11\left(\frac{d^{n-2}}{d\,x^{n-2}}\,\mathrm{y}\,(x)\right)\frac{x^n}{n!}$$

Für den Fall $x = 1$ und unter Ausnutzung der bekannten Anfangswerte für $y(x) = 1$ und $\frac{dy(x)}{dx} = -1$ können die ersten acht Taylorkoeffizienten in rücksubstituierter Form dargestellt werden:

[28] siehe mehr dazu [Bronstein08] auf Seite 463

Ableitung	*Koeffizient*
$y(x)$	1
$\frac{dy(x)}{dx}$	-1
$\frac{d^2y(x)}{dx^2}$	$10\frac{dy(x)}{dx} + 11y(x)$
$\frac{d^3y(x)}{dx^3}$	$111\frac{dy(x)}{dx} + 110y(x)$
$\frac{d^4y(x)}{dx^4}$	$12431\frac{dy(x)}{dx} + 12210y(x)$
$\frac{d^5y(x)}{dx^5}$	$154541971\frac{dy(x)}{dx} + 151782510y(x)$
$\frac{d^6y(x)}{dx^6}$	$23883220952347351\frac{dy(x)}{dx} + 23456768258727210y(x)$
$\frac{d^7y(x)}{dx^7}$	$570408243058643531122685215444411\frac{dy(x)}{dx} + 560223179151189990635941075120710y(x)$

Anhand der überaus großen Zahlen kann bereits ein Trend erahnt werden. Beispielsweise besitzt der elfte Ableitungsterm Faktoren mit einer Länge von 525 Dezimalstellen[29]. Dies liegt weit jenseits der herkömmlich verfügbaren Genauigkeit einer Standardbibliothek zur numerischen Verarbeitung von Zahlen. Dies zeigt somit eine große Schwäche der Taylorreihenmethode. Werden die Faktoren der jeweils verwendeten Ableitungsterme zu groß, so können sie nicht auf Standarddatentypen abgebildet werden. Wird dies dennoch versucht, wird die Zahl bei der Initialisierung auf die verfügbare dezimale Genauigkeit gerundet und ist somit mit einem Fehler behaftet. Dies führt zu einem verfälschten Ergebnis, welches abhängig von der Größenordnung des Fehlers das Ergebnis mehr oder weniger unbrauchbar macht. In Tabelle 2.5 ist das Ergebnis an der Stelle $x = 1$ angegeben. Dabei stellt die analytisch berechnete Methode das korrekte Ergebnis dar. Die Formel zu Berechnung ist auch in [Collatz66] (Seite 49) angegeben und soll hier aus Gründen der Vollständigkeit aufgeführt werden:

$$e^{-x} \tag{2.4.97}$$

Analytisch	*Taylorreihenmethode*
0.367879	$2.7757e + 15$

Tabelle 2.5: Ergebnis der analytischen Lösung und der Taylorreihenmethode an der Stelle $x = 1$

Wie aus Tabelle 2.5 abzulesen ist, haben die beiden Ergebnisse nichts mehr miteinander zu tun. Aus diesem Grund ist bei der Taylorreihenmethode im Vorfeld genau zu prüfen, ob die gewünschte Funktion zur Lösung mit der Taylorreihenmethode geeignet ist.

2.5 Vergrößerung der Integrationsschrittweite

Soll über einen langen Zeitraum integriert werden, sind lange Integrationsschrittweiten wünschenswert. Wird mit einem Standardverfahren integriert, beispielsweise einem Runge-Kutta-Verfahren fester Ordnung, so verschlechtert sich das Ergebnis mit zunehmender Schrittweitenlänge h. Wird andererseits ein Potenzreihenintegrationsverfahren verwendet, so kann die Ordnung der entwickelten Reihe dynamisch gewählt

[29]Der 25.Ableitungsterm besitzt sogar über 34 Millionen Dezimalstellen.

werden. Diese wiederum ist i.d.R. durch den Konvergenzradius[30] begrenzt und verhindert somit ebenfalls große Schrittweitenlängen. In diesem Abschnitt soll eine Möglichkeit aufgezeigt werden, wie mit Hilfe symbolischer Rechnung - angewendet auf die Potenzreihengesetze - eine Vergrößerung der herkömmlich möglichen Integrationsschrittweiten erreicht werden kann. Als Ausgangspunkt hierfür kann das Verfahren der Potenzreihenintegration verwendet werden. Wird eine gewöhnliche Differentialgleichung mit diesem Verfahren gelöst, dann wird für jeden Integrationsschritt eine Potenzreihe der Ordnung m entwickelt. Die so entwickelte Reihe ist somit auf das zu lösende Problem zugeschnitten und kann innerhalb des Konvergenzkreises ausgewertet werden. Nach dem Auswerten der Reihe stehen wieder Anfangswerte zur Verfügung, aus denen unter Verwendung der Differentialgleichung und den Gesetzen der Potenzreihenentwicklung erneut eine Reihe entwickelt werden kann.

Beispiel 2.11 für „Entwicklung einer Reihe zur Lösung der Duffingschen Oszillatorgleichung 3.Ordnung“
Im ersten Schritt ist es erforderlich, anhand vorgegebener Startwerte u und $\dot{u} = \frac{du}{dt}$ eine Reihe 3. Ordnung des Ortes ($u(t+h)$) zu entwickeln. Hier ist t der Startzeitpunkt, h entspricht der Integrationsschrittweite und ω,δ sind Integrationskonstanten. Ferner ist eine zweite Reihe für die erste Ableitung nach der Zeit ($\dot{u}(t+h)$) aufzubauen. Beide Reihen haben folgende Gestalt:

$$\begin{aligned} u(t+h) &= u + \frac{du}{dt}h - \frac{1}{2}h^2\omega^2 u - \frac{1}{2}\epsilon h^2 u^3 \\ \dot{u}(t+h) &= du - \frac{1}{2}\frac{du}{dt}h^2 w^2 - \frac{3}{2}u^2\frac{du}{dt}\delta h^2 - uh\omega^2 - u^3\delta h \end{aligned} \tag{2.5.98}$$

Diese Reihen können nun im Intervall $\lceil -h, h \rceil$ ausgewertet werden. Wird $h = 0$ gesetzt, dann erhält man den Entwicklungspunkt der Reihe. Dieser fällt mit dem jeweiligen Anfangswerten u und $\dot{u}$ zusammen. Nochmaliges Anwenden derselben Methode auf die entwickelte Reihen liefert folgende Reihendarstellung:

$$\begin{aligned} u(t+2h) = {} & \frac{3}{4}u^3(\frac{du}{dt})^2\delta^2h^6 + \frac{3}{2}u^4\frac{du}{dt}\delta^2h^5 - \frac{3}{8}u^7\delta^3h^6 - \frac{1}{2}(\frac{du}{dt})^3\delta h^5 + \frac{1}{4}uh^4\omega^4 - \frac{3}{8}u^3\delta h^6\omega^4 \\ & - \frac{3}{2}u(\frac{du}{dt})^2\delta h^4 + \frac{1}{16}u^3\delta h^8\omega^6 - \frac{3}{4}u^5\delta^2h^6\omega^2 + u + \frac{3}{2}u^2\frac{du}{dt}\delta h^5\omega^2 - 2u^3\delta h^2 \\ & + 2\frac{du}{dt}h - 2uh^2\omega^2 - \frac{3}{8}u^6\frac{du}{dt}\delta^3h^7 + \frac{3}{4}u(\frac{du}{dt})^2\delta h^6\omega^2 - 3u^2\frac{du}{dt}\delta h^3 + u^3\delta h^4\omega^2 \\ & + \frac{1}{16}u^9\delta^4h^8 + \frac{3}{16}u^5\delta^2h^8\omega^4 + \frac{3}{16}u^7\delta^3h^8\omega^2 - \frac{du}{dt}h^3\omega^2 - \frac{3}{4}u^4\frac{du}{dt}\delta^2h^7\omega^2 \\ & - \frac{3}{8}u^2\frac{du}{dt}\delta h^7\omega^4 + \frac{3}{4}u^5\delta^2h^4 \end{aligned} \tag{2.5.99}$$

[30] siehe mehr dazu unter [Rade97]

$$\begin{aligned}\dot{u}(t+2h) = &\frac{15}{16}u^4\frac{du}{dt}\delta^2h^8\omega^4 + \frac{39}{4}u^4\frac{du}{dt}\delta^2h^4 - \frac{5}{2}(\frac{du}{dt})^3\delta h^4 + 3u^5\delta^2h^3 - \frac{9}{4}u^5(\frac{du}{dt})^2\delta^3h^7\\
&- \frac{9}{2}u^5\delta^2h^5\omega^2 + \frac{1}{2}u^9\delta^4h^7 - \frac{9}{4}u^7\delta^3h^5 + \frac{1}{2}u^3\delta h^7\omega^6 - 2u^3\delta h + 9u^3(\frac{du}{dt})^2\delta^2h^5\\
&- \frac{9}{4}u^3\delta h^5\omega^4 + 4u^3\delta h^3\omega^2 + \frac{21}{16}u^6\frac{du}{dt}\delta^3h^8\omega^2 - \frac{39}{8}u^6\frac{du}{dt}\delta^3h^6 - \frac{3}{4}u(\frac{du}{dt})^2\delta h^7\omega^4\\
&+ 9u^2\frac{du}{dt}\delta h^4\omega^2 - 2uh\omega^2 + \frac{du}{dt} + \frac{1}{4}\frac{du}{dt}h^4\omega^4 - 3u^3(\frac{du}{dt})^2\delta^2h^7\omega^2 + \frac{3}{2}u^5\delta^2h^7\omega^4\\
&+ \frac{3}{2}u^7\delta^3h^7\omega^2 + \frac{9}{16}u^8\frac{du}{dt}\delta^4h^8 - 6u^2\frac{du}{dt}\delta h^2 - 6u(\frac{du}{dt})^2\delta h^3 + \frac{9}{4}u^2(\frac{du}{dt})^3\delta^2h^6\\
&- 2\frac{du}{dt}h^2\omega^2 + uh^3\omega^4 + 6u(\frac{du}{dt})^2\delta h^5\omega^2 - \frac{27}{8}u^2\frac{du}{dt}\delta h^6\omega^4 + \frac{3}{4}(\frac{du}{dt})^3\delta h^6\omega^2\\
&- \frac{33}{4}u^4\frac{du}{dt}\delta^2h^6\omega^2 + \frac{3}{16}u^2\frac{du}{dt}\delta h^8\omega^6\end{aligned}$$

Um die etwas umfangreichen Ausdrücke zu verkleinern, werden die Integrationskonstanten $\omega = 1$ *und* $\delta = \frac{1}{100}$ *gesetzt. Dies ergibt:*

$$\begin{aligned}u(t+2h) = &-2uh^2 + \frac{3}{160000}u^5h^8 + 2\frac{du}{dt}h - \frac{1}{200}(\frac{du}{dt})^3h^5 + \frac{3}{16000000}u^7h^8 + u - \frac{3}{800}u^3h^6\\
&- \frac{3}{100}u^2\frac{du}{dt}h^3 + \frac{1}{100}u^3h^4 - \frac{3}{8000000}u^6\frac{du}{dt}h^7 - \frac{du}{dt}h^3 + \frac{3}{400}u(\frac{du}{dt})^2h^6 - \frac{3}{40000}u^5h^6\\
&+ \frac{3}{200}u^2\frac{du}{dt}h^5 - \frac{1}{50}u^3h^2 - \frac{3}{200}u(\frac{du}{dt})^2h^4 + \frac{3}{40000}u^5h^4 + \frac{1}{1600000000}u^9h^8\\
&- \frac{3}{800}u^2\frac{du}{dt}h^7 + \frac{1}{1600}u^3h^8 + \frac{1}{4}uh^4 - \frac{3}{8000000}u^7h^6 + \frac{3}{20000}u^4\frac{du}{dt}h^5 - \frac{3}{40000}u^4\frac{du}{dt}h^7\\
&+ \frac{3}{40000}u^3(\frac{du}{dt})^2h^6\end{aligned} \tag{2.5.100}$$

$$\begin{aligned}\dot{u}(u+2h) = &\frac{9}{40000}u^2(\frac{du}{dt})^3h^6 + \frac{3}{400}(\frac{du}{dt})^3h^6 + \frac{1}{200000000}u^9h^7 + uh^3 + \frac{3}{1600}u^2\frac{du}{dt}h^8 - \frac{1}{40}(\frac{du}{dt})^3h^4\\
&+ \frac{1}{200}u^3h^7 - 2\frac{du}{dt}h^2 + \frac{3}{10000}u^5h^3 - \frac{9}{4000000}u^5(\frac{du}{dt})^2h^7 - \frac{3}{50}u(\frac{du}{dt})^2h^3 - \frac{3}{50}u^2\frac{du}{dt}h^2\\
&- \frac{9}{400}u^3h^5 - \frac{39}{8000000}u^6\frac{du}{dt}h^6 - 2uh + \frac{3}{32000}u^4\frac{du}{dt}h^8 + \frac{1}{4}\frac{du}{dt}h^4 + \frac{9}{100}u^2\frac{du}{dt}h^4\\
&+ \frac{1}{25}u^3h^3 + \frac{3}{20000}u^5h^7 + \frac{du}{dt} - \frac{3}{400}u(\frac{du}{dt})^2h^7 + \frac{9}{1600000000}u^8\frac{du}{dt}h^8 - \frac{27}{800}u^2\frac{du}{dt}h^6\\
&+ \frac{3}{50}u(\frac{du}{dt})^2h^5 - \frac{9}{20000}u^5h^5 + \frac{39}{40000}u^4\frac{du}{dt}h^4 + \frac{3}{2000000}u^7h^7 + \frac{9}{10000}u^3(\frac{du}{dt})^2h^5\\
&- \frac{1}{50}u^3h - \frac{9}{4000000}u^7h^5 - \frac{3}{10000}u^3(\frac{du}{dt})^2h^7 - \frac{33}{40000}u^4\frac{du}{dt}h^6 + \frac{21}{16000000}u^6\frac{du}{dt}h^8\end{aligned}$$

Durch Festlegen der Anfangswerte auf konkrete numerische Werte $u = 1$ *und* $\frac{du}{dt} = 0$ *können die Ausdrücke weiter vereinfacht werden. Einsetzen der Werte ergibt:*

$$\begin{aligned}u(t+2h) &= 1 - \frac{101}{50}h^2 + \frac{1030301}{1600000000}h^8 - \frac{30603}{8000000}h^6 + \frac{10403}{40000}h^4\\
\dot{u}(t+2h) &= \frac{10403}{10000}h^3 - \frac{101}{50}h + \frac{1030301}{200000000}h^7 - \frac{91809}{4000000}h^5\end{aligned} \tag{2.5.101}$$

An dieser Stelle hängen die beiden Ausrücke für Ort und Geschwindigkeit nur noch von der Schrittweite h *ab. Wird nun eine bestimmte Schrittweite gewählt, z.B.:* $h = \frac{1}{100}$ *ergibt dies den*

Abszissenwert an der Stelle $t+\frac{1}{50}$ der Duffingschen Oszillatorgleichung. Für den Fall $h = \frac{1}{100}$ können folgende numerische Werte angegeben werden:

$$\begin{aligned} u(t+2h) &= \frac{1599676804161193879503030 1}{16000000000000000000000000} \\ &= 0.999798 \\ \dot{u}(t+2h) &= -\frac{403979194045903469699}{200000000000000000000000} \\ &= -0.00201989597022951 7348495 \end{aligned} \tag{2.5.102}$$

Ein Vergleich mit der analytischen Lösung 30.Ordnung[31] *ergibt Folgendes:*

Analytische Lösung (30.Ordnung)	$u(t+2h)$
0.9997980069352217684987	*0.99979800*

Hierbei stimmt das Ergebnis mit der analytischen Lösung 30.Ordnung bis zur achten Nachkommastelle überein.

Wie dieses Beispiel demonstriert hat, ist es prinzipiell möglich, eine bestimmte Anzahl von Zwischenschritten zu übergehen. Sollen mehrere Schritte überbrückt werden, so steigt die Anzahl der Terme sehr schnell an, was sich nachteilig auf die Abarbeitungsgeschwindigkeit in der Implementierung auswirkt. Ein wesentlicher Unterschied zu den Standardverfahren ist, dass die hierfür erarbeiteten Ausdrücke exakt auf die Problemstellung zugeschnitten sind und nicht mehr allgemeingültig für eine bestimmte Klasse gewöhnlicher Differentialgleichung geeignet sind. Basierend auf dieser Methode ist es möglich, ein Integrationsverfahren fester Ordnung zu entwerfen, das jeden n.ten Abszissenwert ($n \in N, n > 1$) liefert.

Beispiel 2.12 für „Konstruktion eines analytischen Duffing-Integrators 3.Ordnung“
Hier soll beispielhaft ein Integrationsverfahren zur Lösung der ungedämpften Duffingschen Oszillatorgleichung erstellt werden. Darüber hinaus soll das hier konstruierte Verfahren nur bei jedem zweiten Integrationsschritt einen Abszissenwert erzeugen. Ferner ist in Abb. 2.13 die Arbeitsweise der hier verwendeten Methode zur Konstruktion einer spezialisierten Lösung illustriert. Hierfür wird auf das bereits erarbeitete Ergebnis des vorangegangenen Beispiels zurückgegriffen. Sollen mehrere Schritte nacheinander mit unterschiedlicher Schrittweitenlänge berechnet werden, so müssen die Reihen von $u(t)$, $\frac{du}{dt}$ und von h abhängig sein. Dadurch

[31] Diese wurde in [Mai11] erarbeitet. Im Rahmen dieser Arbeit wurde eine C++-Version der Lösung 30.Ordnung implementiert.

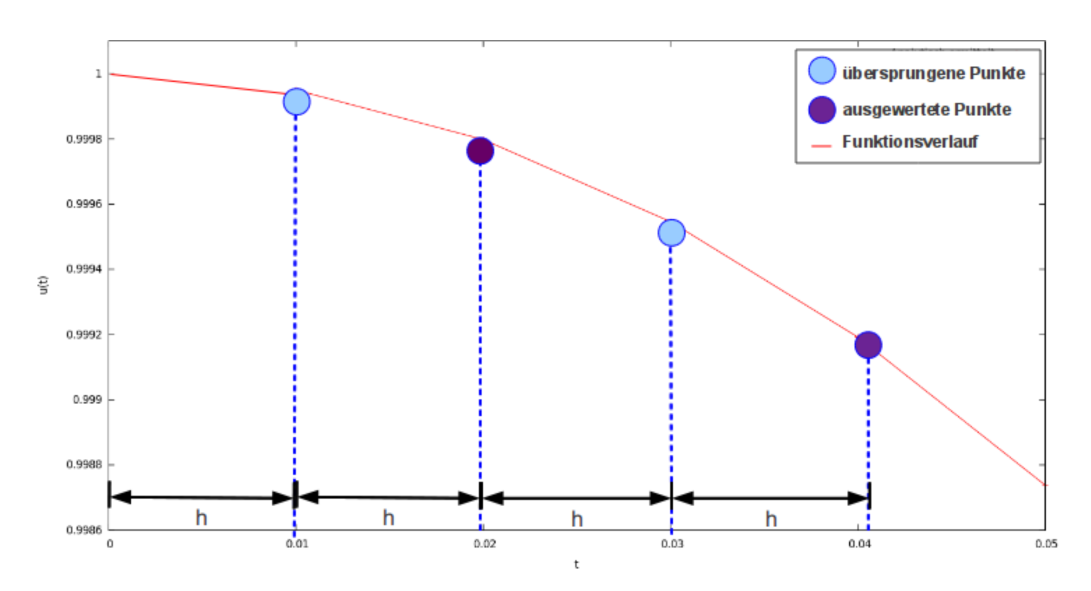


Abbildung 2.13: Veranschaulichung des analytischen Duffing-Integrators 3.Ordnung

ergibt sich folgende formelmäßige Darstellung in Abhängigkeit von der Schrittweite h:

$$\begin{aligned}
u(t+2h) = {} & \frac{3}{400}h^6(\frac{du}{dt})^2u - \frac{3}{8000000}h^6u^7 - \frac{1}{200}h^5(\frac{du}{dt})^3 + \frac{3}{160000}h^8u^5 - \frac{3}{8000000}h^7\frac{du}{dt}u^6 \\
& + 2h\frac{du}{dt} - \frac{1}{50}h^2u^3 - \frac{3}{40000}h^7\frac{du}{dt}u^4 + \frac{3}{16000000}h^8u^7 - \frac{3}{40000}h^6u^5 + \frac{3}{40000}h^6(\frac{du}{dt})^2u^3 \\
& + \frac{3}{200}h^5\frac{du}{dt}u^2 - 2h^2u + \frac{3}{40000}h^4u^5 - \frac{3}{200}h^4(\frac{du}{dt})^2u + \frac{3}{20000}h^5\frac{du}{dt}u^4 + \frac{1}{100}h^4u^3 \\
& - h^3\frac{du}{dt} + \frac{1}{4}h^4u + \frac{1}{1600000000}h^8u^9 + u - \frac{3}{800}h^6u^3 - \frac{3}{800}h^7\frac{du}{dt}u^2 - \frac{3}{100}h^3\frac{du}{dt}u^2 \\
& + \frac{1}{1600}h^8u^3
\end{aligned} \tag{2.5.103}$$

$$\begin{aligned}
\dot{u}(t+2h) = {} & \frac{3}{20000}h^7u^5 + \frac{3}{32000}h^8\frac{du}{dt}u^4 + \frac{du}{dt} + \frac{9}{100}h^4\frac{du}{dt}u^2 - \frac{3}{10000}h^7(\frac{du}{dt})^2u^3 + \frac{3}{2000000}h^7u^7 \\
& + \frac{9}{40000}h^6(\frac{du}{dt})^3u^2 + h^3u - \frac{1}{40}h^4(\frac{du}{dt})^3 + \frac{21}{16000000}h^8\frac{du}{dt}u^6 - \frac{33}{40000}h^6\frac{du}{dt}u^4 - \frac{3}{50}h^2\frac{du}{dt}u^2 \\
& - \frac{9}{20000}h^5u^5 - \frac{39}{8000000}h^6\frac{du}{dt}u^6 + \frac{1}{4}h^4\frac{du}{dt} + \frac{1}{25}h^3u^3 - \frac{9}{4000000}h^5u^7 + \frac{3}{400}h^6(\frac{du}{dt})^3 \\
& - \frac{3}{50}h^3(\frac{du}{dt})^2u - \frac{3}{400}h^7(\frac{du}{dt})^2u + \frac{3}{10000}h^3u^5 - 2hu + \frac{9}{10000}h^5(\frac{du}{dt})^2u^3 - \frac{27}{800}h^6\frac{du}{dt}u^2 \\
& + \frac{9}{1600000000}h^8\frac{du}{dt}u^8 - \frac{9}{400}h^5u^3 + \frac{3}{1600}h^8\frac{du}{dt}u^2 + \frac{39}{40000}h^4\frac{du}{dt}u^4 - 2h^2\frac{du}{dt} + \frac{3}{50}h^5(\frac{du}{dt})^2u \\
& - \frac{9}{4000000}h^7(\frac{du}{dt})^2u^5 - \frac{1}{50}hu^3 + \frac{1}{200000000}h^7u^9 + \frac{1}{200}h^7u^3
\end{aligned}$$

Diese Formeln können nun zur Konstruktion eines Integrationsverfahrens verwendet werden. Sie liefern durch iteratives Einsetzen des jeweils vorangegangen Ergebnisses die Abszissenwerte in äquidistantem Abstand ($2h$). Um herauszufinden welche maximale Schrittweite h verwendet werden kann, wurde das Ergebnis mit der analytischen Lösung (30.Ordnung) verglichen. Der Vergleich der Lösung 3.Ordnung und 4.Ordnung mit der analytischen Lösung

ist in Abb.2.14 dargestellt. Dabei wurden für die jeweiligen Ordnungen Lösungen im Intervall $[0, 0.6]$ *errechnet und diese mit der analytischen Lösung verglichen, indem der absolute Fehler der beiden Lösungen bestimmt wurde. Besonders auffällig beim Vergleich der Lösung 3.Ordnung ist, dass der Fehler nach ca. dreiviertel des untersuchten Intervalls schon bei* 0.001 *liegt. Der Grund für den schnellen Anstieg liegt am verhältnismäßig kleinen Entwicklungsgrad (3.Ordnung) der hier erarbeiteten Lösung. Um dies zu bestätigen wurde eine Lösungsformel 4. Ordnung erarbeitet. Der Vergleich dieser Lösung mit der analytischen Lösung zeigt, dass mit zunehmender Ordnung der Anstieg des absoluten Fehlers abflacht. Mit Hilfe dieser Vergleiche kann der Anstieg des Fehlers für einen Integrationsschritt in Abhängigkeit der verwendeten Schrittweite und Ordnung abgeschätzt werden. Daraus kann in Abhängigkeit des erlaubten Fehlerbudgets eine sinnvolle Schrittweite* h *gewählt werden. Wird nun eine bestimmte Schritt-*

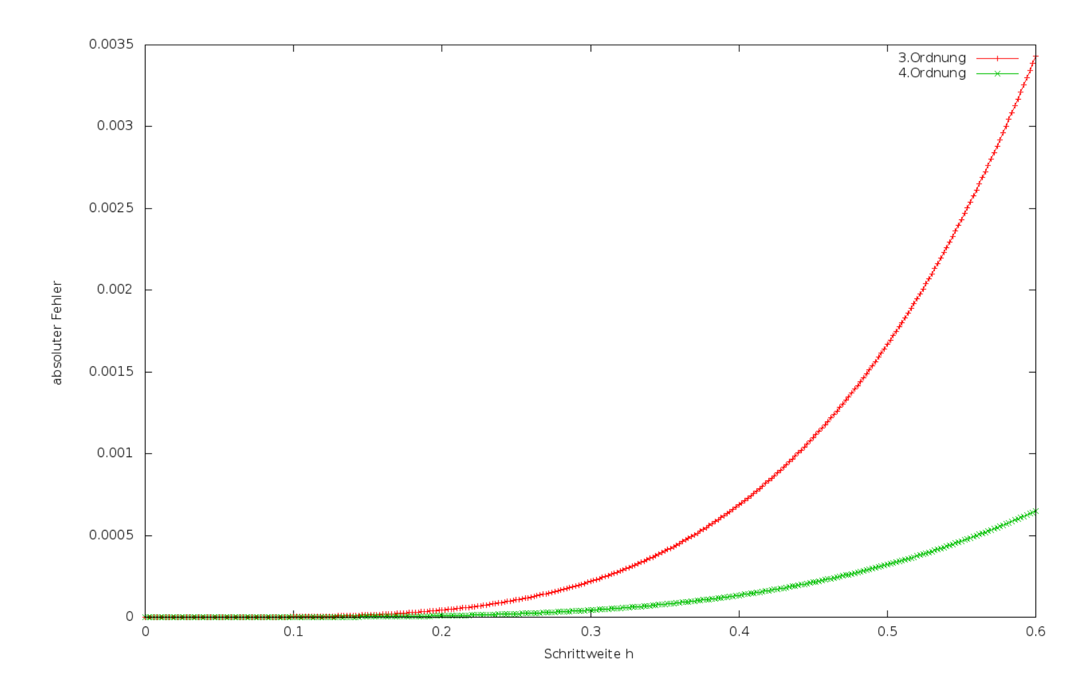


Abbildung 2.14: Vergleich mit analytischer Lösung

weite gewählt, so kann durch mehrfaches hintereinander Ausführen dieser Formeln die gesuchte Lösung berechnet werden.

Um das hier vorgestellte Integrationsverfahren komfortabler zu gestalten, wäre eine automatische Steuerung der Schrittweite wünschenswert. Eine dynamische Änderung der Schrittweite ist in fast allen Fällen nötig, um den resultierenden Gesamtfehler (ohne Berücksichtigung von Eingangs- und Rundungsfehler) zu überwachen. Ansätze zur Implementierung einer adaptiven Schrittweitensteuerung, basierend auf Potenzreihenintegrationsverfahren, ist in [Wanner68] ab Seite 128 zu finden. Dort werden verschiedene Ansätze zur Schrittweitensteuerung diskutiert und es wird eine Formel zu Abschätzung der Schrittweite in Abhängigkeit des vorangegangen Schritts angegeben. Die optimale Schrittweitensteuerung wird ferner in [Morrison62] diskutiert.

2.6 Bewertung der numerischen Integrationsverfahren

In diesem Abschnitt sollen die verwendeten numerischen Integrationsverfahren hinsichtlich ihrer Abarbeitungsgeschwindigkeit, der Anzahl der benötigten Funktionsaufrufe und ihres Fehlerhaushalts anhand einschlägiger Beispiele bewertet werden. Als anfängliches Beispiel soll die ungedämpfte Duffing-Oszillator-Gleichung zur Lösung mit Hilfe einer Auswahl numerische Integrationsverfahren gelöst und die numerisch bestimmten Lösungen mit der analytischen Lösung 30.Ordnung[32] verglichen werden. Im Anschluss wird die Lösung einer Keplerbahn mit verschiedenen Integrationsverfahren berechnet und die Güte der Lösung beurteilt.

Lösungsverfahren	Kategorisierung						
	Schrittweitensteuerung	DG Ordnung		Einschritt-verfahren	Mehrschritt-verfahren	Potenzreihe	Symplektisch
		1	2				
Shampine Gordon (SG)	✓	✓			✓		
Burlisch Stoer (BS)	✓	✓			✓		
Gauss-Jackson (GJ4)			✓	✓			
Runge Kutta 4.order (RK4)		✓		✓			
Runge Kutta 10.order (RK10)		✓		✓			
Dormand-Prince (DOPRI)	✓	✓		✓			
Leap-Frog (LF)		✓		✓			✓
Symplectic (SYMP) 4.-, 6. -and 8.order		✓		✓			✓
Power series (POWSER)		✓		✓		✓	

Abbildung 2.15: Einordnung der implementierten Lösungsverfahren mit deren Eigenschaften

2.6.1 Fehlerbetrachtung

Um für einen bestimmten Fall beurteilen zu können, welches Integrationsverfahren das jeweils bessere ist, wird hier auf die bereits definierten Fehlernormen (siehe Abschnitt 2.1 auf Seite 6) zurückgegriffen. Die numerische Lösung von DG weist jedoch eine spezielle Eigenschaft hinsichtlich ihres Fehlerhaushalts und der Entwicklung des Gesamtfehlers ϵ_g auf. Wird nur der Verfahrens- ϵ_l und Rundungsfehler ϵ_{rd} betrachtet, so kann deren jeweiliger Beitrag zum Gesamtfehler in Beziehung zur gewählten Integrationsschrittweite h gesetzt werden. Dieser Zusammenhang wird in Form einer Fehlerkorrelation in Abb. 2.16 qualitativ dargestellt. Aus dieser Fehlerbetrachtung lässt sich eine optimale Schrittweite h_{opt} ableiten. Diese liegt im Schnittpunkt zwischen der Funktion des Rundungs - und lokalen Diskretisierungsfehlers. Bei Verfahren mit adaptiver Schrittweitensteuerung wird versucht, stets die optimale Schrittweite zu finden um die Fehlerbeiträge möglichst gering zu halten[33]. Einige der Lösungsverfahren besitzen eine feste Konsistenzordnung, wodurch der lokale Diskretisierungsfehler in Abhängigkeit der jeweils gewählten Schrittweite genau bestimmt werden kann. Dabei ist die Konsistenzordnung auf den Vergleich der numerisch bestimmten Lösung

[32] siehe mehr dazu unter [Mai11]
[33] siehe dazu [Speith07]

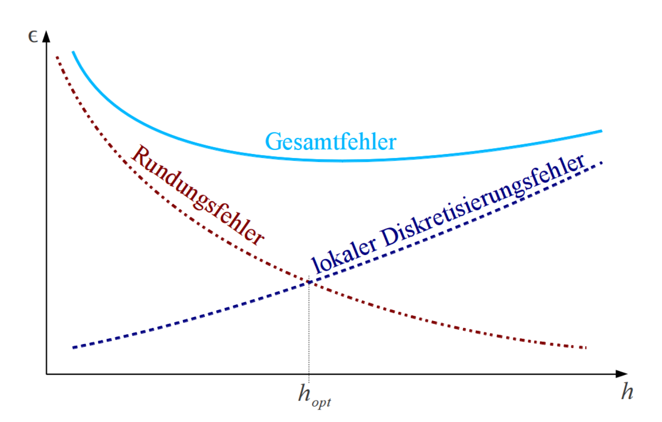


Abbildung 2.16: Fehlerkorrelation von Rundungs - und Diskretisierungsfehler in Abhängigkeit der Integrationsschrittweite

$Y(t+h)$ und der exakten Lösung $y(t+h)$ zum Zeitpunkt $t+h$ zurückzuführen. Beispielsweise ist einem Einschrittverfahren die Konsistenzordnung p zuzuordnen, falls gilt:

$$Y(t+h) - y(t+h) \leq \mathcal{O}(h^p) \tag{2.6.104}$$

2.6.2 Lösung der Duffing-Oszillator-Gleichung

Der ungedämpfte Duffing-Oszillator

Der Duffing-Oszillator kann als Erweiterung zum Harmonischen Oszillator angesehen werden, dem das lineare Hooksche Gesetz zu Grunde liegt und eine kubische Rückstellkraft besitzt. Der ungedämpfte und nicht angeregte Duffing-Oszillator kann formal durch folgende gewöhnliche Differentialgleichung beschrieben werden[34]:

$$\ddot{u} + \omega_0^2 u + \epsilon u^3 = 0 \tag{2.6.105}$$

Beispiel 2.13 für „Untersuchung der Programmlaufzeit“

Als erstes Beispiel soll die Programmlaufzeit gemessen werden. Hierfür wurden folgende AB gewählt:

$$\begin{aligned} u(t_0 = 0.0) &= 1 \\ \dot{u}(t_0 = 0.0) &= 0 \\ \omega &= 1 \\ \epsilon &= \frac{1}{100} \end{aligned} \tag{2.6.106}$$

Die benötigte Zeit Δt, die vergeht, um ein Bewegungsproblem mit Hilfe von numerischer Integration zu bestimmen, hängt von der Plattform ab, auf der die Rechnung durchgeführt wurde. Die Systemeigenschaften der zur Berechnung verwendeten Testplattformen TP1 und TP2 sind im Anhang A.1 auf der Seite 183 beschrieben, sodass alle Testszenarien nachgestellt werden können. Bei den durchgeführten Testrechnungen wurden keinerlei Versuche unternommen, durch Compileroptimierung die Programmlaufzeit zu beeinflussen. Diese und weitere Techniken zu Beschleunigung der Abarbeitungsgeschwindigkeit werden detailliert in Abschnitt 5

[34] siehe dazu [Echardt04]

untersucht. Im Folgenden wird die ungedämpfte Duffing-Oszillatorgleichung im Zeitraum von $t_0 = 0.0$ und $t_{end} = 10.0$ Sekunden mit verschiedenen Lösungsverfahren berechnet und deren benötigte Ausführungszeit beurteilt. Hierbei wird ferner zwischen Verfahren mit und ohne adaptiver Schrittweitensteuerung unterschieden.

[SWS] Schrittweitensteuerung
[RK4] Runge Kutta 4.Ordnung
[MMID] Modified Midpoint Methode
[GJ4] Gauss Jackson 4.Ordnung

Verfahren ohne SWS

In diesem Unterpunkt werden die numerischen Integrationsverfahren RK4, MMID und GJ4 hinsichtlich ihrer benötigten Ausführungszeit einander gegenübergestellt. Hierbei muss eine feste Schrittweite $h = const$ vorgegeben werden. In Tabelle 2.7 auf Seite 46 sind die gemessenen Programmlaufzeiten Δt in Abhängigkeit verschiedener Datentypen, Stellengenauigkeiten und Schrittweiten eingetragen. In der ersten Spalte befindet sich der zur Berechnung verwendete Datentyp. Beim Datentyp long double *handelt es sich um den Standarddatentyp der C-Bibliothek. Der* bigfloat*-Datentyp entstammt der LiDIA-Bibliothek. Mit diesem kann eine beliebige dezimale Genauigkeit gewählt werden. In der zweiten Spalte ist die jeweilige Genauigkeit des Datentyps eingetragen. Die Spalte* Integrator *kennzeichnet das zum Berechnen verwendete Lösungsverfahren und die Spalte h enthält die jeweils verwendete Schrittweite. Die gemessenen Funktionsaufrufe sind in der Spalte 'Aufrufe' eingetragen. Die gemessene Ausführungszeiten auf den entsprechenden TP ist in den Spalten Δt_1 und Δt_2 eingetragen.*

[TP] Testplattform

Datentyp	Genauigkeit	Integrator	h	Aufrufe	Δt_1	Δt_2
long double	19	RK4	6.25e-03	64000	~ 0.126 s	~ 0.175 s
long double	19	RK4	2.44e-06	16384000	~ 36.6 s	~ 44.7 s
bigfloat	19	RK4	6.25e-03	64000	~ 2.3 s	~ 3.0 s
bigfloat	19	RK4	2.44e-06	16384000	~ 10.4 min	~ 12.8 min
bigfloat	200	RK4	6.25e-03	64000	~ 4.7 s	~ 5.3 s
bigfloat	200	RK4	2.44e-06	16384000	~ 18.2 min	~ 22.4 min
long double	19	MMID	6.25e-03	1600	~ 0.007 s	~ 0.003 s
long double	19	MMID	2.44e-06	4096000	~ 6.7 s	~ 7.9 s
bigfloat	19	MMID	6.25e-03	1600	~ 0.053 s	~ 0.053 s
bigfloat	19	MMID	2.44e-06	4096000	~ 1.8 min	~ 2.2 min
bigfloat	200	MMID	6.25e-03	1600	~ 0.101 s	~ 0.102 s
bigfloat	200	MMID	2.44e-06	4096000	~ 3.7 min	~ 4.2 min
long double	19	GJ4	6.25e-03	1616	~ 0.045 s	~ 0.015 s
long double	19	GJ4	2.44e-06	4096016	~ 37.9 s	~ 40.4 s
bigfloat	19	GJ4	6.25e-03	1616	~ 0.272 s	~ 0.237 s
bigfloat	19	GJ4	2.44e-06	4096016	~ 7.8 min	~ 10.1 min
bigfloat	200	GJ4	6.25e-03	1616	~ 0.405 s	~ 0.422 s
bigfloat	200	GJ4	2.44e-06	4096016	~ 13.3 min	~ 18.1 min

Tabelle 2.7: Gegenüberstellung der Lösungsverfahren ohne SWS anhand ihrer Ausführungsgeschwindigkeit

Die Messergebnisse in Tabelle 2.7 lassen folgende Schlussfolgerungen zu:

[DG] Differentialgleichung

- *Unabhängig vom verwendeten Datentyp ist das MMID-Verfahren bezüglich der benötigten Ausführungszeit auf beiden TP das schnellste. Dies liegt an der geringeren Anzahl an Funktionsaufrufen der rechten Seite der DG. Dieses Verfahren wird beim BS-Verfahren als Prädiktor verwendet. Wie im Abschnitt 2.6.1 noch gezeigt wird, erfüllt das MMID-Verfahren nur geringe Genauigkeitsanforderungen und stellt mehr ein Hilfsverfahren dar.*

- *Verglichen mit GJ4, zeigt das RK4-Verfahren ein besseres Laufzeitverhalten.*

- *Es ist nur ein geringer Unterschied in der Ausführungszeit zwischen den verwendeten TP zu erkennen.*

Verfahren mit SWS

Um die implementierten Lösungsverfahren mit SWS aufgrund ihrer benötigten Ausführungsgeschwindigkeit vergleichen zu können, wurde die ungedämpfte Duffing-Oszillatorgleichung mit verschiedenen Datentypen, Fehlertoleranzen und Stellengenauigkeiten berechnet. Der benötigte Zeitaufwand, in Abhängigkeit der verwendeten Konfiguration, ist in der Tabelle 2.8 auf Seite 47 eingetragen. In der Spalte Integrator *ist das zur Berechnung verwendete Lösungsverfahren eingetragen. Die verwendeten Lösungsverfahren benötigen zur automatischen Schrittweitensteuerung die Angabe einer absoluten Fehlertoleranz. Diese ist in der Spalte 'Toleranz' eingetragen. Die Anzahl der benötigten Funktionsaufrufe ist in der Spalte 'Aufrufe' enthalten. Die Spalten Δt_1 und Δt_2 enthalten die gemessene Zeit auf TP1 und TP2 in Sekunden.*

[SG] Shampine Gordon
[BS] Burlisch Stoer

Datentyp	*Genauigkeit*	*Integrator*	*Toleranz*	*Aufrufe*	Δt_1	Δt_2
long double	*19*	*BS*	*1.0e-10*	*718*	$\sim$ *0.027 s*	$\sim$ *0.011 s*
long double	*19*	*BS*	*1.0e-15*	*1346*	$\sim$ *0.037 s*	$\sim$ *0.014 s*
long double	*19*	*BS*	*1.0e-18*	*3812*	$\sim$ *0.054 s*	$\sim$ *0.037 s*
bigfloat	*19*	*BS*	*1.0e-10*	*684*	$\sim$ *0.215 s*	$\sim$ *0.233 s*
bigfloat	*19*	*BS*	*1.0e-15*	*1209*	$\sim$ *0.284 s*	$\sim$ *0.316 s*
bigfloat	*19*	*BS*	*1.0e-18*	*1981*	$\sim$ *0.360 s*	$\sim$ *0.442 s*
bigfloat	*200*	*BS*	*1.0e-30*	*12008*	$\sim$ *6.7 s*	$\sim$ *8.4 s*
bigfloat	*200*	*BS*	*1.0e-40*	*49972*	$\sim$ *27.4 s*	$\sim$ *31.3 s*
bigfloat	*200*	*BS*	*1.0e-50*	*223579*	$\sim$ *1.8 min*	$\sim$ *2.2 min*
bigfloat	*200*	*BS*	*1.0e-60*	*1031313*	$\sim$ *8.4 min*	$\sim$ *10.1 min*
long double	*19*	*SG*	*1.0e-10*	*291*	$\sim$ *0.004 s*	$\sim$ *0.002 s*
long double	*19*	*SG*	*1.0e-15*	*527*	$\sim$ *0.007 s*	$\sim$ *0.004 s*
long double	*19*	*SG*	*1.0e-18*	*1429*	$\sim$ *0.018 s*	$\sim$ *0.014 s*
bigfloat	*19*	*SG*	*1.0e-10*	*291*	$\sim$ *0.065 s*	$\sim$ *0.087 s*
bigfloat	*19*	*SG*	*1.0e-15*	*519*	$\sim$ *0.125 s*	$\sim$ *0.176 s*
bigfloat	*19*	*SG*	*1.0e-18*	*779*	$\sim$ *0.195 s*	$\sim$ *0.261 s*
bigfloat	*200*	*SG*	*1.0e-30*	*9299*	$\sim$ *4.7 s*	$\sim$ *6.1 s*
bigfloat	*200*	*SG*	*1.0e-40*	*37265*	$\sim$ *19.4 s*	$\sim$ *24.5 s*
bigfloat	*200*	*SG*	*1.0e-50*	*195181*	$\sim$ *1.7 min*	$\sim$ *2.3 min*
bigfloat	*200*	*SG*	*1.0e-60*	*1179605*	$\sim$ *10.4 min*	$\sim$ *13.3 min*

Tabelle 2.8: Gegenüberstellung der Lösungsverfahren mit SWS anhand ihrer Ausführungsgeschwindigkeit

Die Betrachtung der Ergebnisse aus Tabelle 2.8 lässt folgende Schlussfolgerung zu:

- *Vergleicht man das SG - und das BS - Verfahren unter Verwendung des* long double *Datentyps, so ist das SG-Verfahren effizienter. Dies lässt sich auch an den benötigten Funktionsaufrufen ablesen (siehe dazu auch 2.6.2).*
- *Wird das SG - und das BS - Verfahren unter Verwendung des* bigfloat*-Datentyps betrachtet, so ist das SG-Verfahren auf beiden TP schneller als das BS-Verfahren. Dies liegt an der verhältnismäßig hohen Anzahl der Funktionsaufrufe des BS - Integrators.*
- *Werden die Datentypen* long double *und* bigfloat *(Genauigkeit 19 Stellen) hinsichtlich der benötigten Rechenzeit betrachtet, so fällt auf, dass der* bigfloat *- Datentyp mehr Rechenzeit benötigt. Dies hat mehrere Gründe:*
  - *Der verwaltungstechnische Mehraufwand der* bigfloat *- Klasse verursacht einen großen Teil der Rechenzeit.*

[TP] Testplattformen

- *Die interne Darstellung unterscheidet sich im Vergleich zu den Standarddatentypen (siehe 3.1.1).*
- *Eine Gleitkommazahl wird innerhalb der 'bigfloat'-Klasse mit Hilfe von Integerzahlen dargestellt. Somit sind für eine Rechenoperation mit dem 'bigfloat' - Datentyp oft viele primitive Rechenoperationen nötig. Dadurch ergibt sich ein entscheidender Geschwindigkeitsnachteil gegenüber den Standarddatentypen. In Abschnitt 4 befindet sich eine Gegenüberstellung von numerischen Bibliotheken mit frei wählbarer Stellengenauigkeit. Darin werden u.a. Leistungsmessungen durchgeführt.*

Beispiel 2.14 für „Anzahl der Funktionsaufrufe“
Die benötigte Anzahl der Funktionsaufrufe ist von entscheidender Bedeutung, falls die Programmlaufzeit eine Rolle spielt. Jede zusätzliche Auswertung der rechten Seite der Differentialgleichung kostet dabei Rechenzeit. Um diese so klein wie möglich zu halten, wird in diesem Abschnitt die automatische Schrittweitensteuerung und deren Verhalten bei einer fest vorgegeben Aufgabenstellung untersucht. Hierfür wird analog zum vorangegangen Beispiel die Bewegungsgleichung des ungedämpften Duffing-Oszillators (siehe Abschnitt2.6.2) zur Lösung verwendet. Dabei wird wieder im Intervall $t \in [0, 10]$ *Sekunden integriert, wobei die geforderte absolute Fehlertoleranz bei den Simulationen im Intervall* $[1.0E-06, 1.0E-15]$ *variiert wurde. In Abbildung 2.17 wurden die benötigten Funktionsaufrufe in Abhängigkeit der jeweils eingestellten und somit geforderten absoluten Fehlertoleranz aufgetragen. Das SG-Verfahren benötigt für diese Aufgabenstellung deutlich weniger Funktionsaufrufe als das BS-Verfahren und ist somit effizienter. Dies bringt besonders bei sehr umfangreichen DG Vorteile in der Laufzeit.*

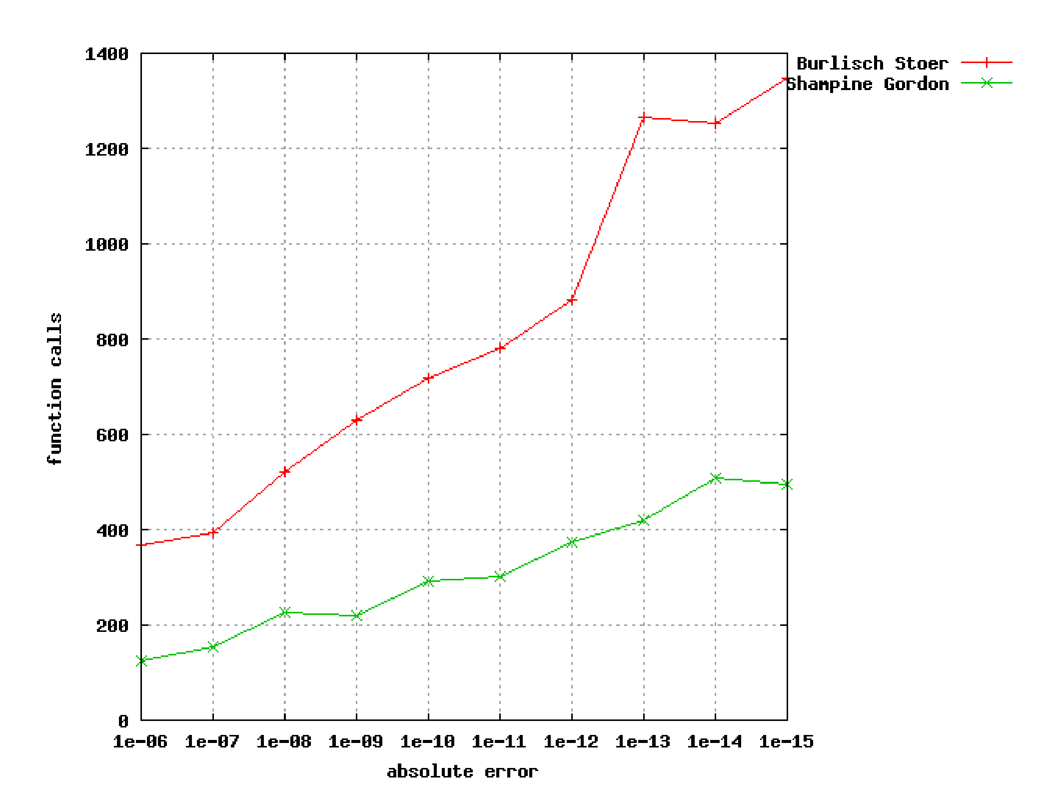


Abbildung 2.17: Benötigte Funktionsaufrufe in Abhängigkeit der verwendeten absoluten Fehlertoleranz der Schrittweitensteuerung

2.6.3 Hin - und Rückrechnung

Die Hin - und Rückrechnung ist ein Mittel zur Visualisierung des Gesamtfehlers ϵ_g^{HR} des jeweiligen numerischen Integrationsverfahrens. Diese ist wie folgt aufgebaut:

1. Hinrechnung:
   Bei diesem Schritt wird von einem Startzeitpunkt t_0 zu einem Endzeitpunkt t_1 integriert.

$$\int_{t_0}^{t_1} f(t, y(t))dt \tag{2.6.107}$$

2. Rückrechnung:
   Bei der Rückrechnung werden die Ergebnisse der Hinrechnung als AB verwendet. Weiter werden hier die Integrationsgrenzen vertauscht, d.h. es wird von t_1 nach t_0 integriert.

[AB] Anfangsbedingung

$$\int_{t_1}^{t_0} f(t, y(t))dt \tag{2.6.108}$$

Der verursachte Gesamtfehler ϵ_g^{HR} der jeweiligen numerischen Lösungsverfahren wird durch Subtraktion des Ergebnisses der Rückrechnung mit den Anfangswerten der Hinrechnung gebildet.

$$\epsilon_g^{HR} = y(x_0) - Y(x_0) \tag{2.6.109}$$

Im Folgenden werden verschiedene Verfahren anhand des Gesamtfehlers gegenübergestellt. Dabei wird zwischen Verfahren mit und ohne SWS unterschieden. Bei den Verfahren ohne SWS wird eine feste Schrittweite ($h = const$) dem Lösungsverfahren vorgegeben.

[SWS] Schrittweitensteuerung

2.6.3.1 Verfahren ohne SWS

In Tabelle 2.9 ist der Gesamtfehler ϵ_g^{HR} der Hin - u. Rückrechnung eingetragen. Hierbei wurden Verfahren ohne automatische SWS untersucht. Die erste Spalte kennzeichnet den verwendeten Datentyp. In der Spalte 'Genauigkeit' ist die Stellengenauigkeit des Datentyps eingetragen. Das verwendete Lösungsverfahren ist in der Spalte 'Integrator' aufgelistet. Bei diesen Verfahren muss eine konstante Schrittweite vorgegeben werden. Die jeweils verwendete Schrittweite ist in der Spalte 'h' eingetragen. Die benötigten Funktionsaufrufe sind der Spalte 'Aufrufe' zu entnehmen.

Datentyp	Genauigkeit	Integrator	h	Aufrufe	ϵ_g^{HR}
long double	19	RK4	6.25e-03	128000	1.36e-17
long double	19	RK4	2.44e-06	32768000	1.23e-27
bigfloat	19	RK4	6.25e-03	128000	1.35e-17
bigfloat	19	RK4	2.44e-06	32768000	1.23e-27
bigfloat	200	RK4	6.25e-03	128000	1.35e-17
bigfloat	200	RK4	2.44e-06	32768000	1.23e-27
long double	19	MMID	6.25e-03	3200	3.53e-3
long double	19	MMID	2.44e-06	8192000	1.41e-6
bigfloat	19	MMID	6.25e-03	3200	3.53e-3

bigfloat	19	MMID	2.44e-06	8192000	1.41e-6
bigfloat	200	MMID	6.25e-03	3200	3.53e-3
bigfloat	200	MMID	2.44e-06	8192000	1.41e-6
long double	19	GJ4	6.25e-03	3232	1.33e-12
long double	19	GJ4	2.44e-06	8192032	2.17e-19
bigfloat	19	GJ4	6.25e-03	3232	1.33e-10
bigfloat	19	GJ4	2.44e-06	8192032	1.19e-27
bigfloat	200	GJ4	6.25e-03	3232	1.33e-10
bigfloat	200	GJ4	2.44e-06	8192032	1.19e-27

Tabelle 2.9: Ergebnisse der Hin - und Rückrechnung der Duffing Oszillatorgleichung unter Verwendung von Lösungsverfahren ohne SWS

Die Messergebnisse aus Tabelle 2.9 lassen folgende Schlussfolgerung zu:

- Werden die Verfahren hinsichtlich ihres Gesamtfehlers ϵ_g^{HR} gegenübergestellt, so ist das GJ4-Verfahren geringfügig besser als das RK4-Verfahren. Das MMID-Verfahren ist dabei deutlich schlechter.
- Werden die benötigten Funktionsaufrufe in die Betrachtung miteinbezogen, so ist das GJ4 - Verfahren am effizientesten.
- Prinzipiell sind diese Verfahren aufgrund fehlender SWS nicht für jede Problemstellung geeignet. Sie werden jedoch oft zur Berechnung eines Anlaufstücks in Prädiktor-Korrektor-Verfahren eingesetzt.

2.6.3.2 Verfahren mit SWS

Die Ergebnisse der Hin - und Rückrechnung mit Mehrschrittverfahren sind in der Tabelle 2.10 auf Seite 52 enthalten. In der ersten Spalte ist der verwendete Datentyp eingetragen. Beim Datentyp 'long double' handelt es sich um den Standarddatentyp der C-Bibliothek. Der 'bigfloat' - Datentyp stammt aus der LiDIA-Bibliothek. Dieser erlaubt eine frei wählbare Stellengenauigkeit. Die jeweils verwendete Genauigkeit ist in der zweiten Spalte angegeben. Die Spalte 'Integrator' kennzeichnet das verwendete numerische Lösungsverfahren. In der Spalte 'Toleranz' ist die verwendete Fehlertoleranz angegeben, welche von der automatischen Schrittweitensteuerung benötigt wird um den lokalen Diskretisierungsfehler bei einem Integrationsschritt zu begrenzen. Die fünfte Spalte enthält die benötigten Funktionsaufrufe für die Hin - u. Rückrechnung. In der sechsten Spalte ist verursachte Gesamtfehler angegeben.

[SG] Shampine Gordon
[BS] Burlisch Stoer

Datentyp	Genauigkeit	Integrator	Toleranz	Funktionsaufrufe	ϵ_g^{HR}
long double	19	BS	1.0e-10	2698	$\sim$ 1.2e-11
long double	19	BS	1.0e-15	2698	$\sim$ 1.2e-16
long double	19	BS	1.0e-18	7376	$\sim$ 8.2e-18
bigfloat	19	BS	1.0e-10	1355	$\sim$ 4.0e-10
bigfloat	19	BS	1.0e-15	2494	$\sim$ 3.5e-15
bigfloat	19	BS	1.0e-18	3789	$\sim$ 1.6e-18

bigfloat	200	BS	1.0e-30	23999	$\sim$ 4.1e-32
bigfloat	200	BS	1.0e-40	99158	$\sim$ 5.9e-44
bigfloat	200	BS	1.0e-50	445981	$\sim$ 1.9e-43
bigfloat	200	BS	1.0e-60	2062146	$\sim$ 1.9e-43
bigfloat	200	BS	1.0e-70	2914140	$\sim$ 1.9e.43
long double	19	SG	1.0e-10	606	$\sim$ 7.2e-11
long double	19	SG	1.0e-15	1215	$\sim$ 1.6e-15
long double	19	SG	1.0e-18	2720	$\sim$ 7.8e-17
bigfloat	19	SG	1.0e-10	606	$\sim$ 7.2e-9
bigfloat	19	SG	1.0e-15	1200	$\sim$ 1.7e-14
bigfloat	19	SG	1.0e-18	1556	$\sim$ 6.2e-18
bigfloat	200	SG	1.0e-30	18586	$\sim$ 2.0e-31
bigfloat	200	SG	1.0e-40	75838	$\sim$ 9.6e-40
bigfloat	200	SG	1.0e-50	386110	$\sim$ 3.0e-50
bigfloat	200	SG	1.0e-60	2331226	$\sim$ 3.2e-60
bigfloat	200	SG	1.0e-70	13954916	$\sim$ 5.2e-70

Tabelle 2.10: Ergebnisse der Hin - und Rückrechnung der Duffing-Oszillatorgleichung unter Verwendung von Lösungsverfahren mit SWS

Aus den Ergebnissen der Hin - und Rückrechnung lassen sich folgende Schlussfolgerungen ableiten:

- Das Anwachsen des Rundungsfehlers kann nicht vermieden werden. Die einzige Möglichkeit liegt darin, die Stellengenauigkeit zu erhöhen. Dies rechtfertigt den Einsatz von numerischen Bibliotheken mit einstellbarer Stellengenauigkeit (siehe Abschnitt 4 auf Seite 87).

- Die verwendete Stellengenauigkeit der C-Standardbibliothek reicht nicht aus, falls über einen längeren Zeitraum integriert wird, da der Gesamtfehler zu stark anwächst.

- Das Lösungsverfahren von SG ist bezüglich der benötigten Funktionsaufrufe effizienter als das BS-Verfahren. Es ist somit bei umfangreichen Differentialgleichungen zu bevorzugen, wie es bei der Berechnung der Kraftfunktion eines Satelliten innerhalb des Gravitationsfeldes der Erde der Fall ist.

- Betrachtet man den Gesamtfehler ϵ_g^{HR} in Abhängigkeit der verwendeten Fehlertoleranz genauer, so fällt auf, dass beim BS-Verfahren dieser zu $\sim$1.9e-43 konvergiert. Beim SG-Verfahren hingegen bleibt ϵ_g unter der gewünschten Fehlertoleranz. Somit ist das BS-Verfahren für Berechnungen mit einer geforderten Fehlertoleranz > 1.0e-50 ungeeignet und das SG-Verfahren zu bevorzugen.

- Bei Toleranzanforderungen $\sim< 1.0e - 43$ ist das BS-Verfahren zu bevorzugen. Es benötigt zur Lösung zwar mehr Funktionsaufrufe als das SG-Verfahren, dafür ist ϵ_g^{HR} kleiner.

Um sicher zu gehen, dass die implementierten Lösungsverfahren korrekt arbeiten, wurden Testrechnungen erstellt und diese von Herrn Mai mit Hilfe seiner in MATHEMATICA[35] implementierten Lösungsverfahren verifiziert. Darüber hinaus wurde die analytische Lösung des ungedämpften Duffing-Oszillators (30.Ordnung[36]) im Rahmen dieser Arbeit in C++ implementiert. Diese wurde u.A. zur Verifikation der Lösungsverfahren herangezogen.

2.6.3.3 Vergleich von Einschritt-Mehrschrittverfahren

Im Folgenden werden Vor - u. Nachteil der Ein - u. Mehrschrittverfahren angegeben.
Vorteile der Einschrittverfahren

- Diese Verfahren sind selbststartend, d.h. sie benötigen keine weiteren Hilfsverfahren zur Berechnung eines Anlaufstücks, wie dies bei Mehrschrittverfahren der Fall ist.
- Die Integrationsschrittweite kann für jeden Integrationsschritt frei gewählt werden.
- Die Schrittweite kann an die Anforderungen der Differentialgleichung angepasst werden.
- Mit *embedded* Runge-Kutta-Verfahren ist es möglich, den lokalen Diskretisierungsfehler zu überwachen und so eine adaptive Schrittweitensteuerung einzuführen, sodass diese bei jeden Integrationsschritt passend gewählt wird[37].

Nachteile von Einschrittverfahren:

- Sie besitzen eine kleinere Konsistenzordnung im Vergleich zu Prädiktor-Korrektor-Verfahren.
- Die Konsistenzordnung ist durch das Verfahren vorgegeben. Soll diese geändert werden, so muss das Verfahren geändert werden.
- Eine Erhöhung der Konsistenzordnung bringt mehr Auswertungen der Differentialgleichung mit sich.
- Um die Genauigkeit zu erhöhen, ist es erforderlich, die Schrittweite h zu verkleinern. Dadurch treten jedoch mehr Rundungsfehler ϵ_r auf.
- Verglichen mit den Prädiktor-Korrektor-Verfahren sind bei diesen Verfahren mehr Auswertungen der Differentialgleichung pro Integrationsschritt nötig.
- Es ist keine brauchbare Fehlerabschätzung angebbar[38].

[35] siehe dazu [Mathematica]
[36] siehe dazu [Mai11]
[37] siehe dazu [Montenbruck05] und [Press92]
[38] siehe dazu [Jordan72]

Vorteile von Mehrschrittverfahren:

- Es sind pro Integrationsschritt nur zwei Auswertungen der Differentialgleichung nötig. Dies ist ein Prädiktor-Schritt und pro Iteration ein Korrektor-Schritt.
- Die Ordnung des Verfahrens kann angepasst werden.
- Durch adaptive Schrittweitensteuerung und integrierte Fehlerabschätzung wird der lokale Diskretisierungsfehler unter einer vorgegebenen Schranke gehalten.
- Treten Diskontinuitäten während der Integration auf, dann beginnt das Verfahren erneut, indem ein neues Anlaufstück berechnet wird.

Nachteile von Mehrschrittverfahren:

- Ergibt sich eine Änderung der Schrittweite, so müssen die vorherigen Punkte neu berechnet werden. Tritt dieser Fall häufig ein, so steigt die Anzahl der Funktionsaufrufe und was sich wiederum negativ auf das Laufzeitverhalten auswirkt.
- Die AW müssen mit Hilfe eines Einschrittverfahrens ermittelt werden.

[AW] Anfangswerte

2.6.4 Integration von Satellitenbahnen

Aufgrund der stetig verbesserten Messgenauigkeit geodätischer Raumverfahren werden auch höhere Anforderung an die numerischen Lösungsverfahren, welche zur Simulation von Satellitenbahnen benötigt werden, gestellt. In diesem Abschnitt soll am Beispiel von zwei Standardproblemen die erreichbare numerische Genauigkeit, in Abhängigkeit verschiedener Randbedingungen, verifiziert werden. Um höhere Genauigkeiten bei der Simulation zu erreichen, können *multiprecision*-Bibliotheken zur Berechnung verwendet werden. Dies ist mit zusätzlichem Rechenzeitaufwand verbunden und soll abgeschätzt werden. Zur Lösung von Differentialgleichungen gibt es eine Vielzahl verschiedener Verfahren, aus denen für diese Untersuchung einige relevante ausgewählt wurden.

Das Keplerproblem
Die folgenden Variablen werden als gegeben vorausgesetzt:

$$\vec{r} = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix}; \quad r = \sqrt{x^2(t) + y^2(t) + z^2(t)};$$

$$\dot{\vec{r}} = \begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{z}(t) \end{pmatrix}; \quad \dot{r} = \frac{dr}{dt} = \frac{2z\left(t\right)\left(\frac{d}{dt}z\left(t\right)\right) + 2y\left(t\right)\left(\frac{d}{dt}y\left(t\right)\right) + 2x\left(t\right)\left(\frac{d}{dt}x\left(t\right)\right)}{2\sqrt{z^2\left(t\right) + y^2\left(t\right) + x^2\left(t\right)}} = \frac{[\vec{r}\dot{\vec{r}}]}{r};$$

$$\mu = GM_{\oplus};$$

dabei ist

- $\vec{r}$ der Ortsvektor im Inertialsystem und r die geometrische Distanz.

- $\dot{\vec{r}}$ der Vektor der Geschwindigkeitskomponenten und $\dot{r}$ die Radialgeschwindigkeit.
- μ die Gravitationskonstante G multipliziert mit der Masse der Erde $M_\oplus$.
- t die Zeit.

Hinweise zur Notation
Der Ausdruck $[\vec{a}\vec{b}]$ entspricht dem Skalarprodukt zweier Vektoren, d.h. $[\vec{a}\vec{b}] := a_0 b_0 + ... + a_n b_n$.
Wobei die Vektoren $\vec{a}$ und $\vec{b}$ aus $n+1$ Komponenten bestehen.
Diese Notation ist insbesondere zur Kontrolle der Dimension größerer Ausdrücke hilfreich.

Die Keplerellipse besitze eine Halbachse von $a = 10000$ Kilometer mit einer Exzentrizität $e = \frac{1}{3}$, wodurch die Umlaufzeit bei etwa zwei Stunden (5614 Sekunden) liegt. Für alle folgenden Berechnungen wurden die Anfangswerte aus Tabelle 2.12 verwendet, welche hierfür aus den Keplerschen-Bahnelementen in kartesische Koordinaten umgewandelt wurden.

Komponente	**numerischer Wert**	**Einheit**
$x(t_0)$	-4461.254589873326	[km]
$y(t_0)$	6652.161968871405	[km]
$z(t_0)$	1371.264327186286	[km]
$\dot{x}(t_0)$	-7.282787778641558	[km/s]
$\dot{y}(t_0)$	-2.280408476437688	[km/s]
$\dot{z}(t_0)$	0.006135775178224878	[km/s]

Tabelle 2.12: Anfangswerte, die zur Berechnung verwendet wurden

Für einen ersten Test der Integrationsverfahren wird im folgenden Beispiel über einen Zeitraum von 5614 Sekunden (ein Umlauf) integriert.

Beispiel 2.15 für „Integration über einen Umlauf (5614 Sekunden)“
Hierfür wurden drei Verfahren mit adaptiver Schrittweitensteuerung ausgewählt: BS, SG und DOP. Bei diesen Verfahren muss eine absolute und relative Fehlerschranke sowie eine definierte Ausgabeschrittweite vorgegeben werden. Die einstellbare Ausgabeschrittweite ist für den Vergleich der verschiedenen Verfahren erforderlich, um Stützstellen zu definierten Zeitpunkten zu erhalten. Ansonsten müssten die jeweiligen Stützstellen nachträglich interpoliert werden. Davon wurde hier abgesehen, da darin zusätzliches Fehlerpotential liegt. Zur Darstellung der Gleitkommazahlen wurden die Standarddatentypen double und long double mit einer Maschinengenauigkeit von 1.0e-16 bzw. 1.0e-19 verwendet. Näheres dazu und weitere Informationen können im Abschnitt 3 nachgeschlagen werden.
In den Tabellen 2.13 und 2.14 sind die Ergebnisse für den Simulationszeitraum eines vollständigen Umlaufs dargestellt und nach Integrationsverfahren sortiert. Darin wird jedem Verfahren eine Spalte zugeordnet, worin die Anzahl der benötigten Funktionsaufrufe C und ein mittlerer Fehler - pro Integrationsschritt - eingetragen ist. Letzterer ist wie folgt definiert:
Falls C Funktionsaufrufe notwendig sind und n absolute Fehlerwerte im Zeitintervall $[t_1, t_n]$ bestimmt wurden, wird $\overline{\epsilon_c}$ wie folgt berechnet:

$$\overline{\epsilon_c} = \frac{1}{C}\sum_{i=1}^{n} \epsilon_{abs}(t_i) \quad (2.6.110)$$

Damit kennzeichnet $\overline{\epsilon_c}$ den mittleren absoluten Fehler, welcher durchschnittlich pro Funktionsaufruf begangen wird. Dies ist somit eine Kennzahl für die jeweilige Güte der durchgeführten Berechnung. Ferner ist jeder Zeile eine absolute Fehlertoleranz zugeordnet (links), die als Vorgabe an das Integrationsverfahren übergeben wurde. Zusätzlich ist die Entwicklung der Funktionsaufrufe sowie der kumulative, absolute Fehler in Abhängigkeit der jeweils vorgegebenen Fehlerschranke in Abb. 2.18 und in Abb. 2.19 illustriert. Dort fällst auf, dass der SG-Integrator sowohl beim double, als auch beim long double Datentyp nahezu gleiche Anzahl Funktionsaufrufe benötigt, jedoch beim long double-Datentyp einen wesentlich geringeren kumulativen Fehler aufweist. Außerdem ist der Funktionsverlauf der Experimente mit dem double bzw. long double-Datentype in vielen Fällen ähnlich.

	BS		***SG***		***DOPRI***	
AbsTol	*C*	$\overline{\epsilon_c}$ $[\frac{mm}{Aufruf}]$	*C*	$\overline{\epsilon_c}$ $[\frac{mm}{Aufruf}]$	*C*	$\overline{\epsilon_c}$ $[\frac{mm}{Aufruf}]$
1E-04	*78626*	*1.25E-05*	*60333*	*0.08355*	*101086*	*5.01E-06*
1E-05	*78626*	*1.25E-05*	*78263*	*0.00230*	*101086*	*5.01E-06*
1E-06	*78626*	*1.25E-05*	*97207*	*1.90E-05*	*101086*	*5.01E-06*
1E-07	*78626*	*1.24E-05*	*114823*	*1.33E-05*	*101086*	*5.01E-06*
1E-08	*78751*	*1.09E-05*	*134199*	*3.54E-06*	*101086*	*5.01E-06*
1E-09	*79744*	*1.55E-05*	*152859*	*1.34E-05*	*101086*	*5.01E-06*
1E-10	*87188*	*9.07E-06*	*170731*	*6.03E-06*	*101086*	*5.01E-06*
1E-11	*111509*	*1.32E-05*	*190261*	*6.41E-06*	*101086*	*5.01E-06*
1E-12	*112694*	*1.21E-05*	*208711*	*4.46E-05*	*101086*	*5.01E-06*
1E-13	*115484*	*1.79E-05*	*226751*	*7.26E-06*	*101086*	*5.01E-06*
1E-14	*133516*	*2.46E-05*	*248573*	*2.13E-06*	*101086*	*5.08E-06*
1E-15	*146110*	*2.28E-05*	*264763*	*7.33E-06*	*101092*	*5.41E-06*

Tabelle 2.13: Mittlerer absoluter Fehler pro Funktionsaufruf und Anzahl benötigter Funktionsaufrufe für den Integrationszeitraum eines Umlaufs unter Verwendung des double-Datentyps

Bei der Berechnung eines Umlaufs ist zu beobachten, dass sich die entsprechenden Fehler in Abhängigkeit des verwendeten Verfahrens unterschiedlich stark akkumulieren. Dennoch kann hier schon ein Trend erkannt werden: Mit zunehmender Integrationszeit werden mehr Funktionsaufrufe benötigt, was neben den Verfahrensfehler auch einen Anstieg des Rundungsfehlers verursacht. Dies soll im nächsten Beispiel, bei dem über einen Integrationszeitraum von einem Tag integriert wird, weiter untersucht werden.

	BS		***SG***		***DOPRI***	
AbsTol	*C*	$\overline{\epsilon_c}$ $[\frac{mm}{Aufruf}]$	*C*	$\overline{\epsilon_c}$ $[\frac{mm}{Aufruf}]$	*C*	$\overline{\epsilon_c}$ $[\frac{mm}{Aufruf}]$
1E-04	*78626*	*7.75E-07*	*60333*	*0.08355*	*101086*	*2.82E-09*
1E-05	*78626*	*7.75E-07*	*78263*	*0.00233*	*101086*	*2.82E-09*
1E-06	*78626*	*7.75E-07*	*97207*	*5.71E-05*	*101086*	*2.82E-09*

1E-07	*78633*	*7.75E-07*	*114823*	*1.41E-06*	*101086*	*2.82E-09*
1E-08	*78751*	*7.61E-07*	*134199*	*8.80E-09*	*101086*	*2.82E-09*
1E-09	*79744*	*7.48E-07*	*152859*	*4.90E-09*	*101086*	*2.82E-09*
1E-10	*87188*	*2.97E-07*	*170731*	*1.16E-08*	*101086*	*2.82E-09*
1E-11	*111405*	*6.87E-09*	*190261*	*1.24E-08*	*101086*	*2.82E-09*
1E-12	*112694*	*8.16E-09*	*208711*	*2.97E-09*	*101086*	*2.82E-09*
1E-13	*115484*	*8.38E-09*	*226725*	*1.61E-08*	*101086*	*2.82E-09*
1E-14	*179348*	*1.88E-09*	*247713*	*3.03E-07*	*101086*	*2.82E-09*
1E-15	*146014*	*1.56E-09*	*273207*	*1.40E-07*	*101086*	*2.82E-09*
1E-16	*146030*	*1.56E-09*	*300821*	*1.48E-08*	*101086*	*2.88E-09*
1E-17	*146152*	*1.79E-09*	*329361*	*1.87E-09*	*101098*	*2.84E-09*
1E-18	*147938*	*2.98E-09*	*355385*	*4.56E-10*	*101266*	*2.83E-09*

Tabelle 2.14: Mittlerer absoluter Fehler pro Funktionsaufruf und Anzahl benötigter Funktionsaufrufe für den Integrationszeitraum eines Umlaufs unter Verwendung des long double-Datentyps

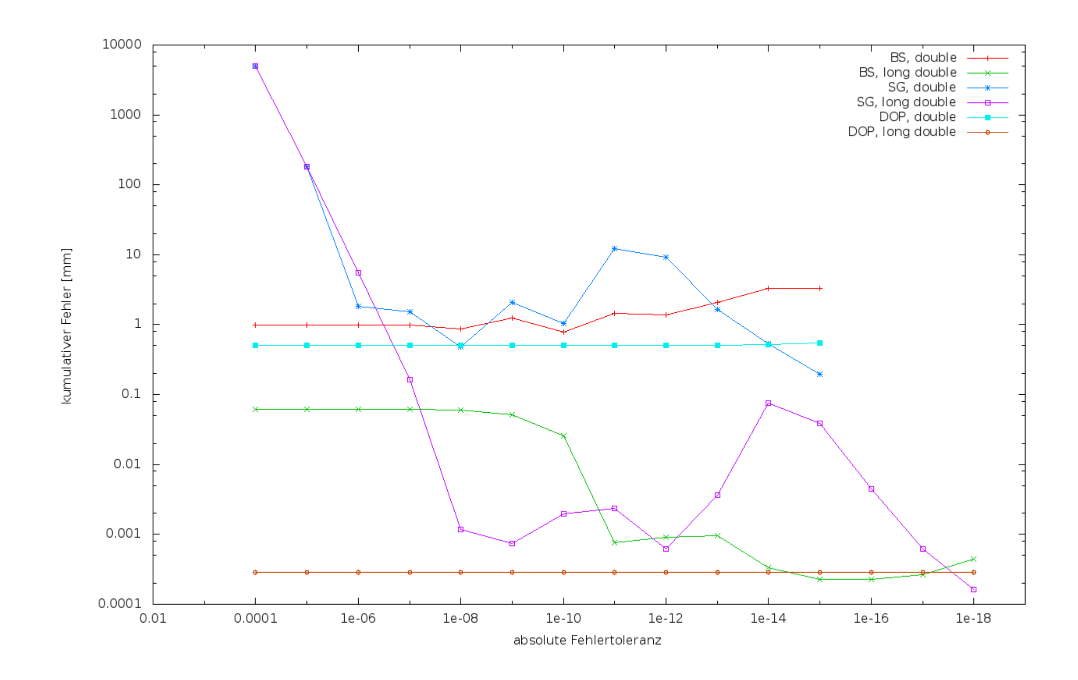


Abbildung 2.18: Kumulativer absoluter Fehler bei einer geforderten Genauigkeit (AbsTol) und einem Umlauf

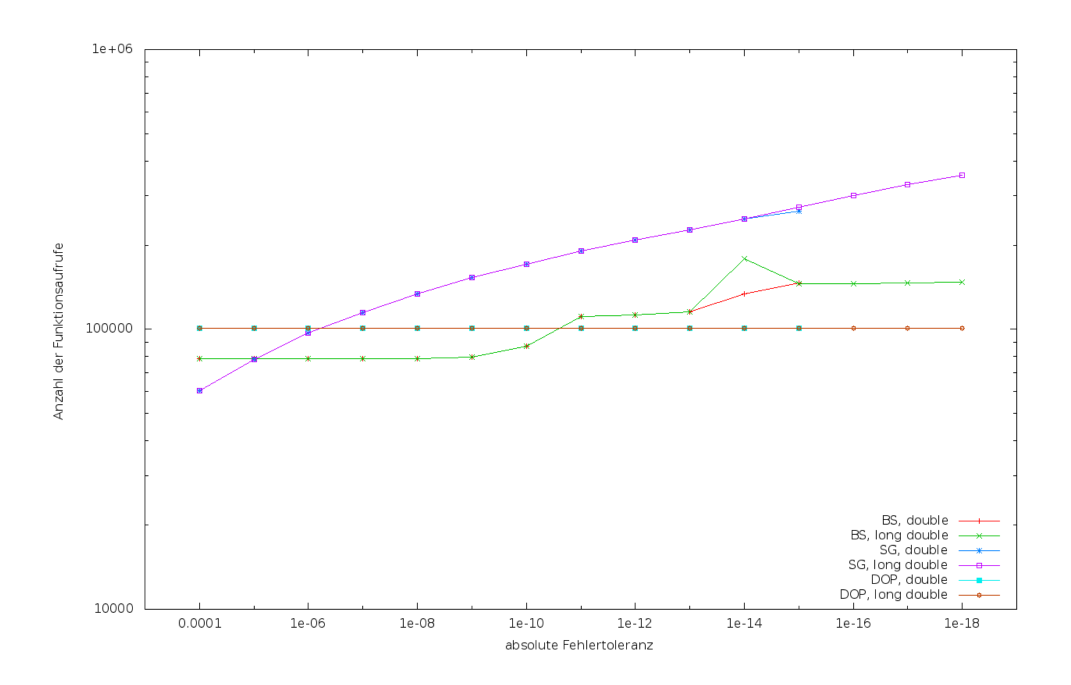


Abbildung 2.19: Anzahl der benötigten Funktionsaufrufe für einen Umlauf in Abhängigkeit der eingestellten Genauigkeit (AbsTol)

Beispiel 2.16 für „Untersuchung zum Simulationszeitraum von einem Tag (15 Umläufe)“
In diesem Beispiel wurde der Integrationszeitraum auf einen Tag erweitert. Dies entspricht bei der hier verwendeten Bahnkonfiguration etwas mehr als 15 Umläufen. Dieses Experiment soll den Verlauf des Fehlers für diese Zeitspanne ermitteln, um die verwendeten Integrationsverfahren besser beurteilen zu können. Betrachtet man den Verlauf des kumulativen Fehlers in Abhängigkeit von der jeweils geforderten Genauigkeit (Abb.2.20) für den Fall des double-*Datentyps, so ist das SG-Verfahren bei der maximal geforderten Genauigkeit von 1.0E-15 das Verfahren mit dem kleinsten Fehler. Aus Tabelle 2.15 ist jedoch abzulesen, dass dafür die meisten Funktionsaufrufe nötig sind. Falls der* long double-*Datentyp verwendet und ebenfalls die maximal geforderte Genauigkeit von 1.0E-18 betrachtet wird, so ist hier das SG-Verfahren bezüglich seiner Fehlerakkumulation das bessere Verfahren. Hervorzuheben ist jedoch hier die vergleichsweise hohe Anzahl von Funktionsaufrufen (siehe Tab. 2.16), die bei umfangreicheren Differentialgleichungen zu Problemen bei der Abarbeitungsgeschwindigkeit führen können. Im Allgemeinen kann für diesen Zeitraum das SG-Verfahren empfohlen werden, falls die Laufzeit eine untergeordnete Rolle spielt, aber auf Genauigkeit Wert gelegt wird. Falls die Programmlaufzeit wichtig ist, scheint das BS-Verfahren ein guter Kompromiss zwischen Aufwand und Genauigkeit zu sein.*

	BS		***SG***		***DOPRI***	
AbsTol	*C*	$\overline{\epsilon_c}$ [$\frac{mm}{Aufruf}$]	*C*	$\overline{\epsilon_c}$ [$\frac{mm}{Aufruf}$]	*C*	$\overline{\epsilon_c}$ [$\frac{mm}{Aufruf}$]
1E-04	*181456*	*6.16E-02*	*152872*	*4.80E-01*	*207376*	*3.53E-04*
1E-05	*181743*	*6.16E-02*	*180358*	*2.19E-02*	*207376*	*3.53E-04*
1E-06	*185611*	*7.82E-02*	*209928*	*2.81E-01*	*207376*	*3.53E-04*
1E-07	*203612*	*3.16E-02*	*238710*	*1.18E-01*	*207376*	*3.53E-04*
1E-08	*232850*	*2.60E-04*	*266388*	*1.19E-01*	*207376*	*3.53E-04*
1E-09	*239696*	*1.36E-04*	*296208*	*3.55E-01*	*207376*	*3.53E-04*

1E-10	*264869*	*9.26E-05*	*326246*	*4.69E-03*	*207376*	*3.53E-04*
1E-11	*366804*	*9.77E-05*	*368889*	*2.35E-02*	*207376*	*3.53E-04*
1E-12	*288726*	*1.10E-04*	*421333*	*1.15E-03*	*207448*	*3.68E-04*
1E-13	*310310*	*9.35E-05*	*470748*	*7.57E-05*	*209878*	*3.51E-04*
1E-14	*362176*	*1.00E-04*	*520312*	*1.86E-05*	*226814*	*1.28E-04*
1E-15	*411008*	*1.32E-04*	*554155*	*1.04E-05*	*263122*	*1.63E-04*

Tabelle 2.15: Mittlerer absoluter Fehler pro Funktionsaufruf und Anzahl benötigter Funktionsaufrufe für den Integrationszeitraum eines Tages unter Verwendung des double-Datentyps

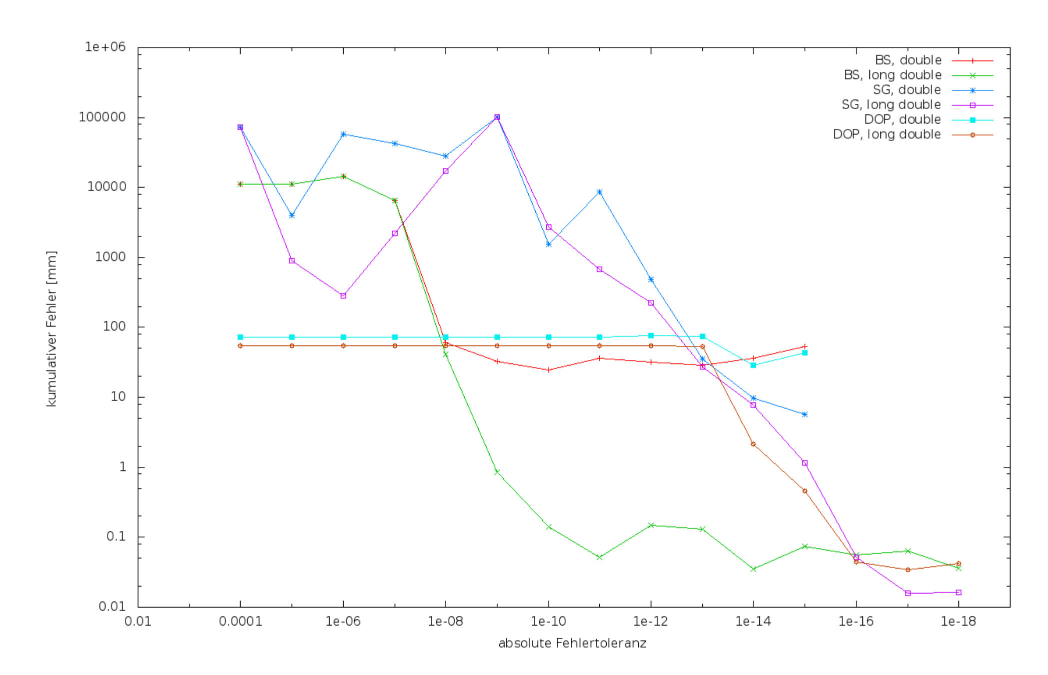


Abbildung 2.20: Kumulativer absoluter Fehler über verlangter Genauigkeit (AbsTol) bei einem Tag

	BS		***SG***		***DOPRI***	
AbsTol	*C*	$\overline{\epsilon_c}$ $[\frac{mm}{Aufruf}]$	*C*	$\overline{\epsilon_c}$ $[\frac{mm}{Aufruf}]$	*C*	$\overline{\epsilon_c}$ $[\frac{mm}{Aufruf}]$
1E-04	*181456*	*6.19E-02*	*152872*	*4.78E-02*	*207376*	*2.61E-04*
1E-05	*181743*	*6.17E-02*	*180358*	*5.00E-03*	*207376*	*2.61E-04*
1E-06	*185611*	*7.83E-02*	*209928*	*1.36E-03*	*207376*	*2.61E-04*
1E-07	*203612*	*3.18E-02*	*238710*	*9.21E-03*	*207376*	*2.61E-04*
1E-08	*232850*	*1.75E-04*	*266386*	*6.46E-02*	*207376*	*2.61E-04*
1E-09	*239696*	*3.57E-06*	*296134*	*3.47E-02*	*207376*	*2.61E-04*
1E-10	*264869*	*5.36E-07*	*324792*	*8.38E-03*	*207376*	*2.61E-04*
1E-11	*298774*	*1.73E-07*	*353847*	*1.89E-03*	*207376*	*2.61E-04*
1E-12	*299726*	*4.98E-07*	*389204*	*5.81E-04*	*207448*	*2.61E-04*

1E-13	*325293*	*4.06E-07*	*432054*	*6.28E-05*	*209878*	*2.55E-04*
1E-14	*453858*	*7.67E-08*	*506577*	*1.55E-05*	*226819*	*9.56E-06*
1E-15	*402584*	*1.82E-07*	*570239*	*2.02E-06*	*262820*	*1.73E-06*
1E-16	*406520*	*1.35E-07*	*623165*	*8.36E-08*	*315775*	*1.41E-07*
1E-17	*411610*	*1.54E-07*	*676719*	*2.38E-08*	*408142*	*8.37E-08*
1E-18	*459779*	*8.14E-08*	*736662*	*2.20E-08*	*566842*	*7.51E-08*

Tabelle 2.16: Mittlerer absoluter Fehler pro Funktionsaufruf und Anzahl benötigter Funktionsaufrufe für den Integrationszeitraum eines Tages unter Verwendung des long double-Datentyps

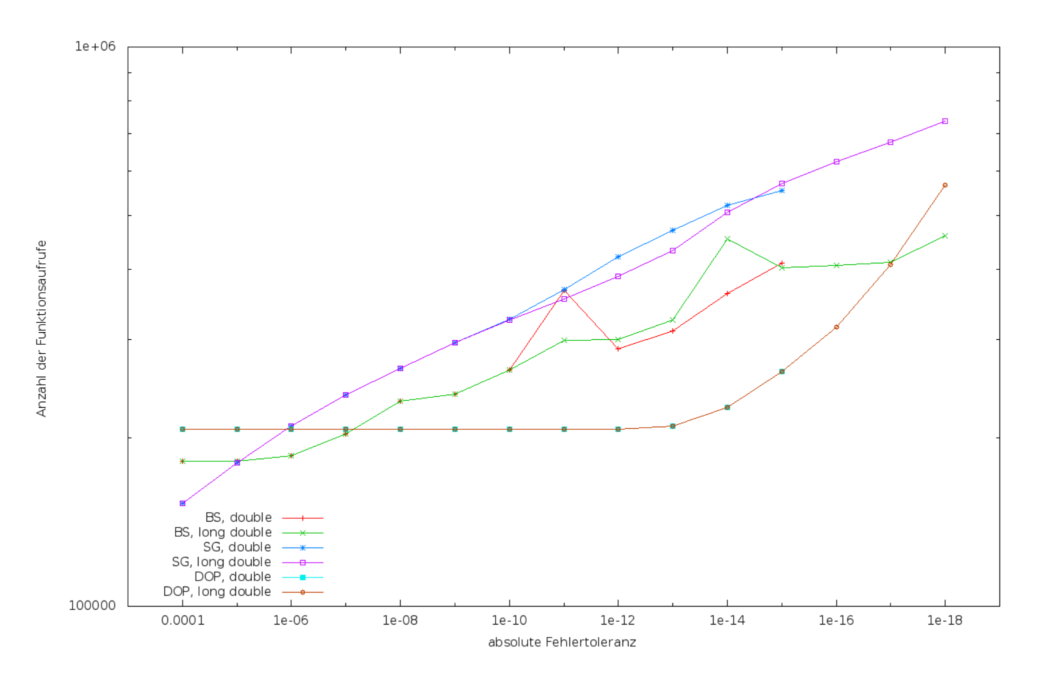


Abbildung 2.21: Anzahl der Funktionsaufrufe bei der Integration über einen Zeitraum von einem Tag

Beispiel 2.17 für „Untersuchung zum Simulationszeitraum von einer Woche (ca. 108 Umläufe)“
In diesem Experiment wird wieder das Keplerproblem integriert, wobei der Integrationszeitraum auf eine Woche ausgedehnt wird. Ziel ist es, die Stabilität der verwendeten Integratoren für diesen relativ langen Zeitraum zu beurteilen.

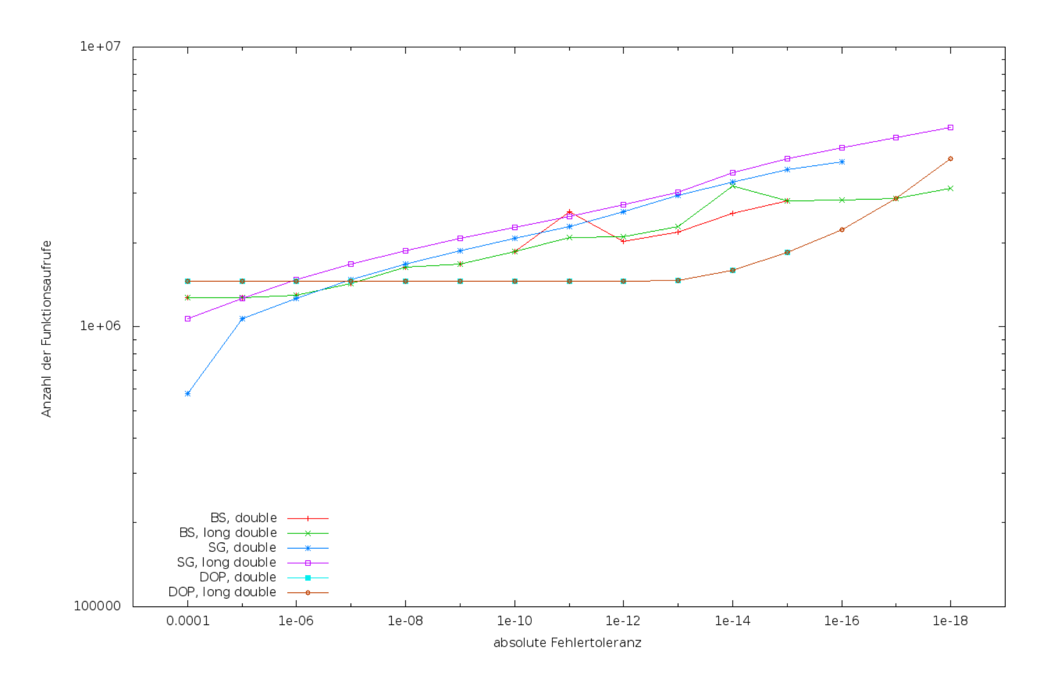


Abbildung 2.22: Anzahl der Funktionsaufrufe bei der Integration über einen Zeitraum von 7 Tagen Integrationszeit

Betrachtet man die Anzahl der benötigten Funktionsaufrufe (Abb. 2.22), so fällt auf, dass analog zu den beiden vorangegangen Experimenten das SG-Verfahren, unabhängig vom verwendeten Datentyp, am meisten Funktionsaufrufe benötigt. Hinsichtlich tatsächlich erzielter Genauigkeit ist das SG-Verfahren sowohl beim double *als auch beim* long double*-Datentyp, das Verfahren mit der kleinsten Fehlerakkumulation. Jedoch ist allgemein betrachtet, die Größenordnung des kumulativen Fehlers sogar beim SG-Verfahren im Millimeterbereich.*

	BS		**SG**		**DOPRI**	
AbsTol	*C*	$\overline{\epsilon_c}$ $[\frac{mm}{Aufruf}]$	*C*	$\overline{\epsilon_c}$ $[\frac{mm}{Aufruf}]$	*C*	$\overline{\epsilon_c}$ $[\frac{mm}{Aufruf}]$
1E-04	1270096	7.01E-01	577051	2.26E+01	1451536	2.35E-02
1E-05	1272483	6.19E-01	1071082	1.08E+00	1451536	2.35E-02
1E-06	1300456	6.28E-01	1263552	1.45E+01	1451536	2.35E-02
1E-07	1429507	1.36E+00	1470654	9.84E+01	1451536	2.35E-02
1E-08	1630881	1.96E-02	1672054	6.09E+01	1451536	2.35E-02
1E-09	1679656	8.31E-03	1865802	1.91E+01	1451536	2.35E-02
1E-10	1859907	7.40E-03	2074674	2.29E-01	1451536	2.35E-02
1E-11	2572245	3.49E-03	2284608	5.74E-01	1451536	2.35E-02
1E-12	2019722	9.68E-03	2585813	1.04E-01	1452268	2.31E-02
1E-13	2179906	7.79E-03	2953427	1.46E-03	1469740	2.26E-02
1E-14	2540070	1.08E-02	3301553	9.49E-04	1592658	9.62E-03
1E-15	2818892	1.02E-02	3649210	2.88E-05	1849884	8.59E-03

Tabelle 2.17: Mittlerer absoluter Fehler pro Funktionsaufruf und Anzahl benötigter Funktionsaufrufe für den Integrationszeitraum von einer Woche unter Verwendung des double-Datentyps

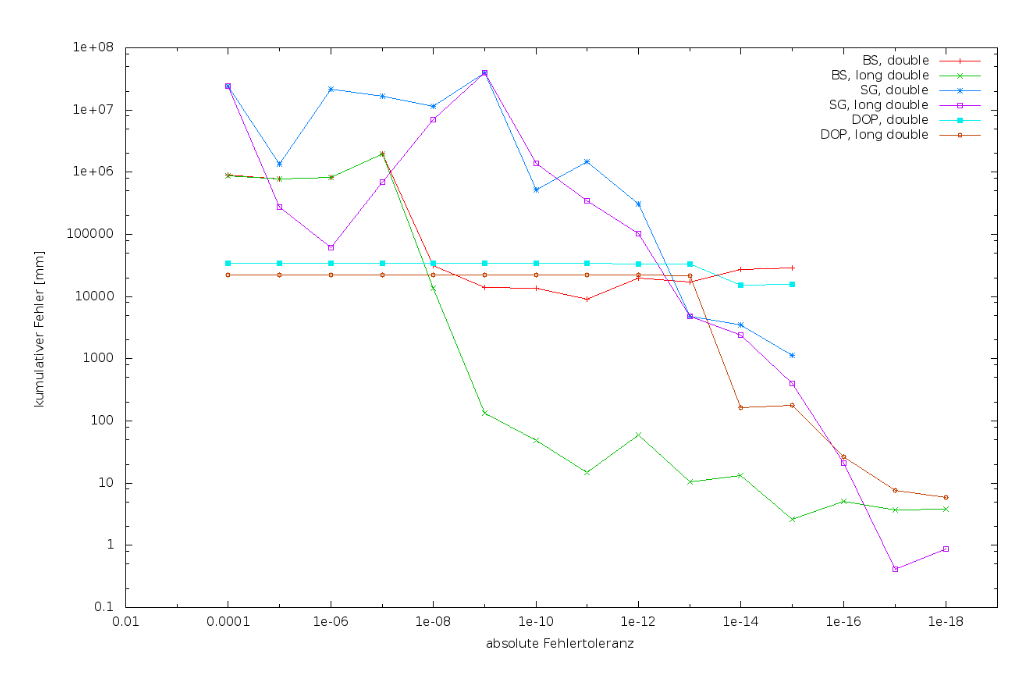


Abbildung 2.23: Kumulativer absoluter Fehler bei einer geforderten Genauigkeit (AbsTol) und 7 Tagen Integrationszeit

	BS		***SG***		***DOPRI***	
AbsTol	*C*	$\overline{\epsilon_c}$ [$\frac{mm}{Aufruf}$]	*C*	$\overline{\epsilon_c}$ [$\frac{mm}{Aufruf}$]	*C*	$\overline{\epsilon_c}$ [$\frac{mm}{Aufruf}$]
1E-04	*1270096*	*6.90E-01*	*1071082*	*2.26E+01*	*1451536*	*1.52E-02*
1E-05	*1272483*	*6.07E-01*	*1263552*	*2.19E-01*	*1451536*	*1.52E-02*
1E-06	*1300456*	*6.40E-01*	*1470654*	*4.20E-02*	*1451536*	*1.52E-02*
1E-07	*1429507*	*1.37E+00*	*1672054*	*4.09E-01*	*1451536*	*1.52E-02*
1E-08	*1630881*	*8.38E-02*	*1865798*	*3.77E+00*	*1451536*	*1.52E-02*
1E-09	*1679656*	*8.03E-05*	*2074076*	*1.91E+01*	*1451536*	*1.52E-02*
1E-10	*1859933*	*2.63E-05*	*2274624*	*6.10E-01*	*1451536*	*1.52E-02*
1E-11	*2089943*	*7.02E-06*	*2478495*	*1.39E-01*	*1451536*	*1.52E-02*
1E-12	*2102126*	*2.79E-05*	*2728139*	*3.78E-02*	*1452268*	*1.52E-02*
1E-13	*2287440*	*4.56E-06*	*3029628*	*1.61E-03*	*1469794*	*1.48E-02*
1E-14	*3185356*	*4.16E-06*	*3551863*	*6.83E-04*	*1592783*	*1.02E-04*
1E-15	*2818784*	*9.33E-07*	*3997471*	*9.89E-05*	*1846973*	*9.53E-05*
1E-16	*2845240*	*1.79E-06*	*4369635*	*4.82E-06*	*2221644*	*1.19E-05*
1E-17	*2877952*	*1.28E-06*	*4745828*	*8.79E-08*	*2879014*	*2.64E-06*
1E-18	*3122878*	*1.24E-06*	*5167584*	*1.67E-07*	*4003223*	*1.46E-06*

Tabelle 2.18: Mittlerer absoluter Fehler pro Funktionsaufruf und Anzahl benötigter Funktionsaufrufe für den Integrationszeitraum von einer Woche unter Verwendung des long double-Datentyps

2.6.5 Fazit zur Integration von Satellitenbahnen mit herkömmlicher Genauigkeit

Die durchgeführten Experimente haben gezeigt, dass bei einer relativ einfachen Problemstellung, wie dem Keplerproblem, die Resultate der hier untersuchten Integrationsverfahren durchaus große Unterschiede aufweisen. Es wurde gezeigt, dass bei einem Integrationszeitraum von einer Woche eine Ungenauigkeit von einem Millimeter[39] nur mit einem der drei verwendeten Verfahren eingehalten werden konnte. In einer umfangreicheren Simulation einer Satellitenbahn, bei der beispielsweise Kräfte wie der Strahlungsdruck der Sonne und die Erdanziehungskraft miteinbezogen werden, fallen signifikant mehr Rechenoperationen pro Integrationsschritt an. Diese resultieren in einem Anstieg des Rundungsfehlers, der das Ergebnis zusätzlich verschlechtert. Somit sind die hier durchgeführten Experimente als Minimalbeispiele zu betrachten, deren Ergebnis durch mehr Rechenoperation verschlechtert und keinesfalls verbessert wird. Soll über einen Zeitraum von einer Woche hinaus integriert werden und sind höhere Anforderungen an die Genauigkeit gestellt, so empfiehlt sich der Einsatz von Datentypen mit mehrfacher Genauigkeit.

[39] siehe dazu auch in Anhang F.3 auf Seite 219, wo die Entwicklung des absoluten Fehlers über den Simulationszeitraum von sieben Tagen aufgetragen ist. Hierbei wird ebenfalls bestätigt, dass nach einer Woche Simulationsdauer höchstens ein Millimeter Genauigkeit gefordert werden kann.

Kapitel 3

Implementierungsaspekte

Schwerpunkt des Kapitels:

In diesem Abschnitt wird die interne Darstellung von Gleitkommazahlen und ihr Rundungsverhalten diskutiert. Darauf aufbauend wird neben einigen anderen Beispielen eine Rundungsfehleranalyse der Duffing- Oszillatorgleichung durchgeführt. Schließlich soll kurz auf die Konzepte der objektorientierten Programmierung und die programmiertechnische Umsetzung eingegangen werden. Diese Hinführung zum Thema der objektorientierten Programmierung wird benötigt, um während der Entwicklung der Lösungsverfahren getroffene Designentscheidungen besser nachvollziehen zu können. Einige Hauptmerkmale daraus sind:

- Das Klassenkonzept
- Das Konzept der Kapselung
- Das Vererbungskonzept
- Die Operatorüberladung
- Die Template-Programmierung

Im Anschluss wird mit Hilfe von Quelltextauszügen die programmiermäßige Vorgehensweise zur Lösung einer Differentialgleichung schrittweise erklärt.

3.1 Gleitkommazahlen

Die Entwicklung des Gleitkommazahlenformats ist historisch gesehen die Antwort auf die Frage, welches Maschinenformat für die Darstellung reeller Zahlen notwendig ist, um alle vier Grundrechenarten (Addition, Subtraktion, Multiplikation und Division) möglichst performant durchführen zu können[1]. Die interne Binärdarstellung erlaubt die effiziente Durchführung von Addition und Subtraktion. Die Multiplikation und Division kann mit Hilfe von logarithmischer Zahlendarstellung auf Addition und Subtraktion zurückgeführt werden, wie folgendes Beispiel der Verknüpfung zweier Gleitkommazahlen a und b zeigt:

$$log(a * b) = log(a) + log(b)$$
$$log(a/b) = log(a) - log(b)$$

Damit ist es prinzipiell möglich, sämtliche mathematischen Transformationen darauf aufzubauen. Jedoch müsste hierbei unnötigerweise zwischen binärer und dezimaler Zahlendarstellung gewechselt werden, was sich negativ auf die Programmlaufzeit auswirken würde. Aus diesem Grund erfand Konrad Zuse im Jahr 1935 die Gleitkommazahlendarstellung[2]. Daraus entstand 1985 der IEEE-754-1985[3] Standard für Gleitpunktarithmetik. Aus diesem ging eine überarbeitete und erweiterte Fassung hervor, der IEEE-754-2008 Standard für Gleitpunktarithmetik. Dieser definiert u.a. einen Standard für Dezimalarithmetik und Gleitpunktarithmetik mit 128-Bit Mantisse. Es wird das Buch von [Knuth98] (ab Seite 214) empfohlen, dort wird die Rechnung mit Gleitkommazahlen und deren interne Darstellung diskutiert.

3.1.1 Interne Darstellung einer Gleitkommazahl nach IEEE

[IEEE 754] IEEE Standard for Binary Floating-Point Arithmetic for microprocessor systems (ANSI/IEEE Std 754-1985)

Die Norm IEEE 754 definiert die Darstellung einer Gleitkommazahl x in einem Rechner wie folgt:

- $\underbrace{x}_{Gleitkommazahl} = \underbrace{s}_{Vorzeichen} \cdot \underbrace{m}_{Mantisse} \cdot \underbrace{b^e}_{Basis^{Exponent}}$
- Vorzeichen s:
  Bei normalisierten Gleitkommazahlen nach IEEE 754 ist die Basis $b = 2$. Das Vorzeichen $s = (-1)^S$ benötigt ein Bit zur Speicherung. Ist $S = 0$, so wird eine positive Zahl und für $S = 1$ eine negative Zahl dargestellt.
- Mantisse m:
  Die Mantisse m enthält die Ziffern der Gleitkommazahl. Je mehr Ziffern abgespeichert werden, desto größer wird die Genauigkeit. Die Anzahl p der Mantissenbits drückt also aus, wie exakt die Zahl approximiert wird. Dieser Parameter wird entweder direkt oder in Form der kleinsten darstellbaren Zahl ϵ_{fp} angegeben. Diese Zahl beschreibt den kleinsten darstellbaren Wert, welcher zur Zahl 1 hinzuaddiert werden kann und ein von 1 verschiedenes Ergebnis liefert[4].

[1] entnommen [Huckle06], Seite 56
[2] siehe mehr dazu http://www.zuse.org
[3] siehe dazu [IEEE-754-1985]
[4] siehe [Press92]

- Basis b:
  Seit der Norm für Gleitkommazahlen IEEE 754, wird in den meisten Computern fast ausschließlich die Basis $b = 2$ verwendet.

- Exponent e:
  Der Exponent kennzeichnet die Position des Kommas der Gleitkommazahl. Damit ist auch die Größenordnung der Gleitkommazahl festgehalten. Da die Anzahl r der Exponentenbits limitiert ist, wird durch den Exponenten der maximal darstellbare Zahlenbereich begrenzt. Die verfügbaren Zahlenbereiche der C-Standarddatentypen sind aus der Tabelle 4.2 ersichtlich.

3.1.2 Standardrundung nach IEEE-754

Die Mantisse einer normalisierten Gleitkommazahl kann wie folgt dargestellt werden:

$$m = 1.b_1 b_2 b_3 ... b_{p-1} | b_p \tag{3.1.1}$$

Entscheidend für die Rundung sind die Bits $b_{p-1}|b_p|b_{p+1}$. Je nach Zustand (0 oder 1) wird entschieden, ob auf - oder abgerundet wird.

b_{p-1}	b_p	b_{p+1}	Aktion
0 oder 1	0	0 oder 1	abrunden
0	1	0	abrunden
1	1	0	aufrunden
0	1	1	aufrunden
1	1	1	aufrunden

Im Standard sind außerdem noch drei weitere Rundungsverfahren definiert (*floor*, *ceil* und *round to zero*). Diese haben, verglichen mit der Standardrundung, einen größeren Fehler und sind deswegen für Standardfälle nicht zu empfehlen. Eine Fehlerabschätzung hierfür ist in [Huckle06] auf Seite 64 zu finden. Des Weiteren werden die Standards IEEE-754-1985, IEEE-854-1987 und IEEE-754-2008 in [Muller10] (Kapitel 3, ab Seite 55) mit ihren speziellen Eigenschaften genau beschrieben.

3.1.3 Bekannte Probleme bei der Rechnung mit Gleitkommazahlen

Gleitkommazahlen werden verwendet, um Zahlen darzustellen und diese mathematischen Transformationen weiter zu verarbeiten. In diesem Abschnitt soll kritisch hinterfragt werden, ob der internen Abbildung von Gleitkommazahlen auf die interne Struktur des Rechners und der darauf aufbauenden Verarbeitungsroutinen blind vertraut werden kann. Des Weiteren sollen einige Schwachpunkte aufgezeigt werden, um den Blick für potentielle Fehler[5] in Programmen zu schärfen.

Prominente Beispiele zu Fehlerfällen, welche durch falsche Verwendung von Gleitkommazahlen verursacht wurden, gibt es genügend. Hier einige Auszüge zu verschie-

[5] Hierzu hat sich ein eigener Forschungszweig entwickelt und es existieren bereits einige Programme, mit denen die Implementierung von Gleitkommaoperationen auf einer bestimmten Hardwareplattform überprüft werden können. Hierzu hat u.a. [Verdonk04] ein Programm entwickelt, welches die IEEE-754-Kompatibilität prüft.

denen Disziplinen der Wissenschaft:

- *Due to an error caused by a homemade data-analysis program, on page 1875, Geoffrey Chang and his colleagues retract three Science papers and report that two papers in other journals also contain erroneous structures.*[6]
- *... was done in by some homebrew software which swapped two columns of data. In a structure this large and complicated, you can have such disruptive things happen and still be able to settle down on a final protein picture - it's just that it'll be completely wrong*[7]
- *... have lead to the determination and publication of incorrect structures.*[8]
- *...An in house data reduction program introduced a change in sign for anomalous differences...We very sincerely regret the confusion that these papers have caused and, in particular, subsequent research efforts that were unproductive as a result of our original findings.*[9]

Die aufgeführten Beispiele zeigen, dass diese Probleme keineswegs der Vergangenheit angehören. Diese Erkenntnis zeigt die Notwendigkeit genauer Betrachtung numerisch berechneter Ergebnisse und deren darauf folgende Interpretation. Ferner muss sichergestellt sein, dass die verwendete Hardwareplattform korrekte Ergebnisse erzeugt. Dieses Misstrauen ist gerechtfertigt, betrachtet man die lange Liste der prominenten Hardware - u. Software-Fehler der letzten Jahrzehnte:

- Bei der ersten Generation von Intel Pentium Prozessoren (1987) trat bei der Division bestimmter Zahlen ein Fehler auf. Beispielsweise liefert die Berechnung des Bruchs

$$8391667/12582905$$

anstatt $0.666910...$ das falsche Ergebnis: $0.666869...$.

[CAS] ComputerAlgebraSystem

- Das Computeralgebrasystem MAPLE[10] hatte mit der Version 7.0 Probleme bei der korrekten Berechnung des Ausdrucks:

$$\frac{1001!}{1000!}$$

Dieser Ausdruck wurde fälschlich zu 1 und nicht zu 1001 vereinfacht[11].

Integerzahlen werden auch als Ganzzahlen bezeichnet.

- Mit der vorausgegangenen Version wurde eine lange Integerzahl falsch eingelesen. Die Zahl

$$21474836480413647819643794$$

[6] siehe dazu [Miller06]
[7] siehe dazu [Lowe07]
[8] siehe dazu [Jeffrey07]
[9] siehe dazu [Kavanagh10]
[10] siehe dazu [Maple]
[11] Der Fehlerbericht zu diesem Programmierfehler kann unter folgendem Weblink eingesehen werden: http://mathforum.org/kb/message.jspa?messageID=1568841, Download am 12.03.2012

wurde durch einen Fehler zu

$$413647819643790 +' - - .(- - .($$

konvertiert.

Die aufgeführten Beispiele sollen lediglich die Schwächen heutiger Softwareroutinen aufzeigen und konsequenterweise dazu verleiten, numerisch bestimmte Ergebnisse sorgfältig zu prüfen. Weitere Beispiele dieser Art sind u.a. in [Muller10] und [Huckle06] zu finden.
Ist sichergestellt, dass die verwendete Hardwareplattform vertrauenswürdige Ergebnisse liefert, so muss anhand entsprechender Kontrollrechnungen die Software auf ihre Richtigkeit überprüft werden. Dies erfolgt in der Regel durch *unit*-Tests. Dabei werden für jede der verwendeten Funktionen Testfälle programmiert, welche das Verhalten der Funktion aufgrund unterschiedlicher Eingabeparameter überprüfen. Diese werden gesammelt und bieten die Möglichkeit, das gesamte Programmsystem oder auch nur ein bestimmtes Modul durch den Aufruf eines Befehls zu verifizieren. In der industriellen Softwareentwicklung ist diese Vorgehensweise bereits Standard und sollte ebenfalls im wissenschaftlichen Bereich Einzug halten. Eine ausführliche Beschreibung zum Softwaretest mit *unit*-Tests findet man in [Martin09] (Kapitel 9, ab Seite 159).
Viele der oben aufgeführten Ursachen hätten durch umfangreicheres Testen vermieden werden können. Die häufigsten Fehler können wie folgt kategorisiert werden: Division durch Null, Über- bzw. Unterläufe oder Fehler, die durch Summierung von Rundungsfehlern entstehen.
Prinzipiell stellen Datentypen für Gleitkommazahlen Zahlen nicht als kontinuierliche Zahlenfolge dar, sondern als Exponent, multipliziert mit einer arithmetischen Reihe. Dies bedeutet, es gibt eine untere und eine obere Schranke für die kleinste (`FLOAT_MIN`) bzw. die größte darstellbare Gleitkommazahl (`FLOAT_MAX`) pro Datentyp. Durch das Design der Gleitkommazahlen-Darstellung ist vorgegeben, dass zwischen zwei aufeinander folgenden Gleitkommazahlen eine Lücke besteht. Soll nun eine Zahl dargestellt werden, die im Bereich der Lücke liegt, so wird diese auf die nächst nähere Zahl gerundet.

C-Quelltext	Ausgabe
float f=0.1;	
printf(„%.10f“, f);	*0.1000000015*

Tabelle 3.2: Implizites Runden durch Basiswechsel

Im C-Quelltext (siehe Tabelle 3.2) wird die Zahl 0.1 (Basis 10) in die Binärdarstellung umgewandelt. Gerade diese Zahl kann nicht auf die interne Darstellung abgebildet werden, woraufhin implizit gerundet werden muss. Der dadurch entstehende Rundungsfehler kann durch Rücktransformation auf die Basis 10 sichtbar gemacht werden (siehe rechte Spalte von Tabelle 3.2). Die Differenz zwischen der ursprünglichen Zahl mit dem Ergebnis der Rücktransformation wird als impliziter Rundungsfehler bezeichnet. Hierbei sind zwei wichtige Eigenschaften zu betrachten:

1. Dieser Fehler kann nicht vorausberechnet werden, da dieser von der jeweiligen Implementierung abhängig ist.

2. Das Ergebnis nach der Rücktransformation kann entweder unter oder über dem anfänglichen Wert liegen, d.h. die Ausgabe nach der Rücktransformation könnte entweder 0.1000000015 oder auch 0.99999998 lauten.

Um dennoch auf das korrekte Ergebnis zu kommen, kann ein Trick angewandt werden. Dieser ist jedoch nur möglich, falls das Ergebnis und das Rundungsverhalten vorher bekannt ist. Dabei wird ein Korrekturwert angebracht, welcher das Ergebnis der Rücktransformation entsprechend korrigiert. Das daraus errechnete Ergebnis stimmt exakt mit dem ursprünglichen Wert 0.1 überein (siehe Tabelle 3.3).

Zeilennummer	C-Quelltext	Ausgabe
1	*float f(0.1);*	
2	*float fHelper(-0.0000000015);*	
3	*printf("%.10f\n",f+fHelper);*	*0.1000000000*

Tabelle 3.3: Anbringen einer Korrektur zur Minimierung des Rundungsfehlers

In der ersten Zeile wird die Variable `f` mit dem Wert `0.1` initialisiert. Anschließend erfolgt in der zweiten Zeile die Initialisierung einer Hilfsvariablen mit einem Korrekturwert. In der letzten Zeile wird vor der Ausgabe der Korrekturwert angebracht und auf die Standardausgabe geschrieben. Diese Methode funktioniert nur für diese Aufgabenstellung und diesen Wert und ist somit nicht allgemein gültig. Das Beispiel soll vielmehr zeigen, dass implizites Runden fester Bestandteil von Gleitkommazahlen ist. Auch durch Vergrößerung der Mantisse kann dieses Problem nicht behoben werden, sondern es wird lediglich verlagert. Abhilfe hierfür bietet die Rationalarithmetik bzw. die Dezimalarithmetik. Ersteres vermeidet Gleitkommazahlen, indem jede Zahl aus Zähler und Nenner dargestellt wird. Diese bestehen aus ganzen Zahlen (Integer) und werden nicht gerundet. Leider ist diese Variante für numerische Integrationsverfahren ungeeignet, da schon nach wenigen Zeitschritten ungemein große Bruchterme entstehen, die nicht weiter vereinfacht werden können. Die Dezimalarithmetik geht dabei einen anderen Weg: Sie führt keinen Basiswechsel zwischen dezimaler und binärer Darstellung durch und vermeidet somit implizites Runden bei der Initialisierung.
In der Literatur [Huckle06] finden sich weitere prominente Beispiele zu Softwarefehlern und die dadurch verbundenen Auswirkungen. Außerdem wird der Artikel von [Merali10] empfohlen. Darin befinden sich nennenswerte Statistiken zu einer Umfrage, bei der ca. 2000 Wissenschaftler aus unterschiedlichen Fachgebieten befragt wurden. Hier ein inhaltlicher Auszug: 38% der Wissenschaftler verbringen ein Fünftel ihrer Arbeitszeit mit Softwareentwicklung, jedoch haben nur 47% davon das nötige Hintergrundwissen zum Softwaretest. Außerdem finden 66% der Wissenschaftler eine Ausbildung in Softwareentwicklung als überflüssig. Ferner wird darin beklagt, dass aufgrund mangelnder Dokumentation die Wahrscheinlichkeit für Folgefehler enorm ansteigt.

3.1.4 Antwort auf die Frage: Warum werden nicht ausschließlich Gleitkommazahlen verwendet?

Der Grund dafür liegt im Design der Gleitkommazahlen und dem daraus resultierenden Fehlerpotential. Gleitkommazahlen stellen, anders als Ganzzahlen, nur einen approximierten Wert dar. Dies führt dazu, dass das Assoziativgesetz ([Bronstein08], Seite 327, Gleichung 5.8a) bei Gleitpunktoperation nicht gilt. Folglich können die beiden Rechnungen ein unterschiedliches Ergebnis erzeugen:

Für alle $b! = 0$

Gleitkommaarithmetik	Ganzzahlarithmetik
$(a+b)+c \neq a+(b+c)$	$(a+b)+c = a+(b+c)$
$(a*b)/b \neq a;$	$(a*b)/b = a$

Aus diesem Grund ist es empfehlenswert, Gleitkommazahlen nur dort einzusetzen, wo es unbedingt nötig ist.

3.1.5 Rundungsfehleranalyse (Epsilontik)

Durch Analysieren des Rundungsfehlers kann das Fehlerverhalten eines Verfahrens beurteilt und die Größenordnung des Fehlers abgeschätzt werden. Dazu ist anfänglich eine Begriffsdefinition nötig, welche aus [Huckle06] entnommen wurde:

Beispiel 3.1 für „Realisierung einer Gleitkommaoperation“

Für jede Gleitkommaoperation $\circ \, \epsilon\{+,-,*,/\}$ *mit dem die Gleitkommazahlen* $a_1, a_2 \in M$ *verknüpft werden, gilt folgende Rechenregel:*

$$a_1 \circ_M a_2 = rd(a_1 \circ a_2) = (a_1 \circ a_2)(1+\epsilon_\circ) \tag{3.1.2}$$

wobei M *die Menge aller Gleitkommazahlen ist. Der relative Fehler ist deshalb stets beschränkt durch* $|\epsilon_\circ| \leq \epsilon$*. Die Abschätzung des Rundungsfehlers mit der Maschinengenauigkeit* ϵ *ist von zentraler Bedeutung und ermöglicht nicht nur die Rundungsfehleranalyse einer einzelnen Rechenoperation, sondern auch die Analyse der Fehlerfortpflanzung durch eine Folge von Rechenoperationen.*

Im Folgenden soll die Analyse der Fehlerfortpflanzung für den allgemeinen Fall erarbeitet werden. Der Ausdruck $a_1 \odot a_2$ bedeute, dass zwei Gleitkommazahlen mit einer Rechenoperation ($\{+,-,*,/\}$) verknüpft werden. Aus dieser Verknüpfung kann konsequenterweise ein Rundungsfehler hervorgehen. Außerdem bedeute die Verknüpfung $a_1 \circ a_2$, dass hier kein Rundungsfehler entsteht. Im folgenden Beispiel soll eine allgemeine Formel zu Abschätzung des Rundungsfehlers erarbeitet werden. Diese hängt weder von der Art der Verknüpfung, noch von der Anzahl der hintereinander ausgeführten Verknüpfungen ab.

3.1.6 Allgemeine Fehlerfortpflanzung bei Gleitkomma-Operationen

Es sollen n Gleitkommazahlen $a_1, ..., a_n$ arithmetisch miteinander verknüpft werden, wobei $n \in N$ ist. Die Verknüpfung von n Gleitkommazahlen erfordert $n-1$ arithme-

tische Operationen. Das kann wie folgt dargestellt werden:

$$r_{n-1} = a_1 \circ a_2 \circ ... \circ a_n \tag{3.1.3}$$

Zur besseren Veranschaulichung der einzelnen Verknüpfungen werde das folgende Schema eingeführt, wobei mit r_i für $i \in N$ die jeweiligen Zwischenresultate bezeichnet werden:

$$\begin{aligned} r_1 &= a_1 \odot a_2 \\ r_2 &= r_1 \odot a_3 \\ r_3 &= r_2 \odot a_4 \\ &... \\ r_{n-1} &= r_{n-2} \odot a_n \end{aligned} \tag{3.1.4}$$

Nimmt beispielsweise n den Wert 3 ein, so ergibt sich folgendes Schema:

$$\begin{aligned} r_1 &= a_1 \odot a_2 \\ r_2 &= r_1 \odot a_3 \end{aligned} \tag{3.1.5}$$

Aus der Erkenntnis des vorangegangen Beispiels kann hierfür der Rundungsfehler entwickelt werden:

$$\begin{aligned} r_2 &= r_1 \odot a_3 \\ r_2 &= (r_1 \circ a_3)(1 + \epsilon_2) \\ r_2 &= ((a_1 \odot a_2) \circ a_3)(1 + \epsilon_2) \\ r_2 &= ((a_1 \circ a_2)(1 + \epsilon_1) \circ a_3)(1 + \epsilon_2) \\ r_2 &= a_1 \circ a_2 \circ a_3 + (a_1 \circ a_2)\epsilon_1 + (a_1 \circ a_2 \circ a_3)\epsilon_2 + (a_1 \circ a_2)\epsilon_1\epsilon_2 \end{aligned} \tag{3.1.6}$$

Nach [Huckle06] können die Terme höherer Ordnung $\epsilon_1\epsilon_2$ vernachlässigt werden, da hierbei Folgendes gilt:

$$|\epsilon_1|, |\epsilon_2| \leq \epsilon \tag{3.1.7}$$

Somit sind Terme höherer Ordnung kleiner als ϵ^2 und werden wegen $\epsilon^2 \ll \epsilon \ll 1$ vernachlässigt. Hierbei wird ϵ als Maschinengenauigkeit bezeichnet. Somit kann der erarbeitete Zusammenhang ohne die Fehlerterme höherer Ordnung vereinfacht dargestellt werden:

$$r_2 = a_1 \circ a_2 \circ a_3 + (a_1 \circ a_2)\epsilon_1 + (a_1 \circ a_2 \circ a_3)\epsilon_2 \tag{3.1.8}$$

Aus dem Beispiel für drei Verknüpfungen kann der allgemeinere Zusammenhang für die Verknüpfung von N Variablen wie folgt dargestellt werden.

$$\begin{aligned} r_{n-1} = a_1 \circ a_2 \circ ... \circ a_n& \\ + \epsilon_1(a_1 \circ a_2)& \\ + \epsilon_2(a_1 \circ a_2 \circ a_3)& \\ + ...& \\ + \epsilon_{n-1}(a_1 \circ ... \circ a_n)& \end{aligned} \tag{3.1.9}$$

Dabei sei angemerkt, dass jeder der einzelnen Fehlerterme durch die vorangegangenen Verknüpfungen beeinflusst wird. Im Folgenden sollen für einige ausgewählte Fälle die Formeln zu Abschätzung des Rundungsfehlers erarbeitet werden. Hierbei wird der absolute und relative Fehler als Gütekriterium herangezogen.

Beispiel 3.2 „Rundungsfehleranalyse“
Es soll folgender Fall untersucht werden:

$$\frac{(a * b)}{c} \tag{3.1.10}$$

mit der Einschränkung $a, b, c \in N$ *mit* $c \notin 0$*. Die Anzahl der Verknüpfungen ist zwei, d.h. es werden drei Gleitkommazahlen miteinander verknüpft und somit ist* $n = 3$*. Daraus ergibt sich durch Einsetzen der Verknüpfungen in Reihenfolge der Verarbeitung* $(*, /)$*, folgender Term:*

$$\begin{aligned} r_2 = &\frac{(a * b)}{c} \\ &+ \epsilon_1(a * b) \\ &+ \epsilon_2(\frac{(a * b)}{c}) \end{aligned} \tag{3.1.11}$$

Der relative Fehler, mit $r^* = \frac{(ab)}{c}$ *als Erwartungswert ist somit:*

$$\begin{aligned} \epsilon_{rel} &= \frac{\epsilon_{abs}}{r^*} = \frac{|r^* - r_2|}{r^*} \\ &= \frac{|\frac{(ab)}{c} - (\frac{(ab)}{c} + \epsilon_1(ab) + \epsilon_2\frac{(ab)}{c})|}{\frac{(ab)}{c}} \\ &= \frac{|-(\epsilon_1(ab) - \epsilon_2\frac{(ab)}{c})|}{\frac{(ab)}{c}} \\ &= \epsilon_1 c + \epsilon_2 \end{aligned} \tag{3.1.12}$$

Hieraus ist abzulesen, dass ϵ_1 *ausschließlich durch den numerischen Wert der Variable* c *verstärkt wird und dass die numerischen Werte der Variablen* a *und* b *keinen Einfluss auf den Rundungsfehler haben.*

Bei der vorausgegangen Betrachtung des Rundungsfehlers wurde davon ausgegangen, dass die Anfangswerte mit keinem Eingangsfehler behaftet sind. Ist dies nicht der Fall, so müssen die jeweiligen Anfangswerte mit einem Fehlerterm $(1+\epsilon_{a_i})$ mit $i \in N$ versehen werden. Dies soll für $n = 3$ demonstriert werden. Hierfür wird auf die bereits erarbeitete allgemeine Formel zur Bestimmung des Rundungsfehlers zurückgegriffen. Zunächst ohne Eingangsfehler:

$$\begin{aligned} r_2 = &a_1 \circ a_2 \circ a_3 \\ &+ \epsilon_1(a_1 \circ a_2) \\ &+ \epsilon_2(a_1 \circ a_2 \circ a_3) \end{aligned} \tag{3.1.13}$$

Wird der Eingangsfehler an jeder Variablen angebracht, kann der Ausdruck wie folgt

erweitert werden:

$$\begin{aligned} r_2 = a_1(1+\epsilon_{a_1}) \circ a_2(1+\epsilon_{a_2}) \circ a_3(1+\epsilon_{a_3}) \\ + \epsilon_1(a_1(1+\epsilon_{a_1}) \circ a_2(1+\epsilon_{a_2})) \\ + \epsilon_2(a_1(1+\epsilon_{a_1}) \circ a_2(1+\epsilon_{a_2}) \circ a_3(1+\epsilon_{a_3})) \end{aligned} \tag{3.1.14}$$

Vereinfachung und Umsortierung ergibt:

$$\begin{aligned} r_2 = a_1 \circ a_2 \circ a_3 + a_1\epsilon_{a_1} + a_2\epsilon_{a_2} + a_3\epsilon_{a_3} \\ + \epsilon_1(a_1 \circ a_2) + \epsilon_1(a_1\epsilon_{a_1} \circ a_2\epsilon_{a_2}) \\ + \epsilon_2(a_1 \circ a_2 \circ a_3) + \epsilon_2(a_1\epsilon_{a_1} \circ a_2\epsilon_{a_2} \circ a_3\epsilon_{a_3}) \end{aligned} \tag{3.1.15}$$

Werden analog zum vorangegangenen Vorgehen die Terme höherer Ordnung vernachlässigt, so kann der Ausdruck weiter vereinfacht werden durch:

$$\begin{aligned} r_2 = a_1 \circ a_2 \circ a_3 + a_1\epsilon_{a_1} + a_2\epsilon_{a_2} + a_3\epsilon_{a_3} \\ + \epsilon_1(a_1 \circ a_2) \\ + \epsilon_2(a_1 \circ a_2 \circ a_3) \end{aligned} \tag{3.1.16}$$

Aus dieser Darstellung kann eine allgemein Formel für n Verknüpfungen abgeleitet werden:

$$\begin{aligned} r_{n-1} = a_1 \circ a_2 \circ ... \circ a_n \\ + a_1\epsilon_{a_1} + a_2\epsilon_{a_2} + ... + a_n\epsilon_{a_n} \\ + \epsilon_1(a_1 \circ a_2) \\ + \epsilon_2(a_1 \circ a_2 \circ a_3) \\ + ... \\ + \epsilon_{n-1}(a_1 \circ ... \circ a_n) \end{aligned} \tag{3.1.17}$$

Der relative Rundungsfehler ϵ_{rel} mit r^* als Erwartungswert kann allgemein angegeben werden:

$$\begin{aligned} \epsilon_{rel} &= \frac{\epsilon_{abs}}{r^*} \\ &= \frac{|r^* - r_{n-1}|}{r^*} \\ &= \frac{1}{r^*}(a_1\epsilon_{a_1} + a_2\epsilon_{a_2} + ... + a_n\epsilon_{a_n} \\ &+ \epsilon_1(a_1 \circ a_2) \\ &+ \epsilon_2(a_1 \circ a_2 \circ a_3) \\ &+ ... \\ &+ \epsilon_{n-1}(a_1 \circ ... \circ a_n)) \end{aligned} \tag{3.1.18}$$

Daraus ist abzulesen, dass der Eingangsfehler ϵ_{a_i}, abhängig vom Wert des Gesamtresultats, verstärkt wird. Die Eingangsfehler können beispielsweise durch vorangegangene Rechnungen oder Messfehler in die Rechnung eingebracht worden sein. Dieser

Ausdruck kann nun noch in Eingangs - u. Rundungsfehleranteile aufgeschlüsselt werden:

$$\begin{aligned}\epsilon_{rel} = &\underbrace{\frac{1}{r^*}(a_1\epsilon_{a_1} + a_2\epsilon_{a_2} + ... + a_n\epsilon_{a_n})}_{Eingangsfehler} \\ &+ \frac{1}{r^*}(\\ &+ \epsilon_1(a_1 \circ a_2) \\ &+ \epsilon_2(a_1 \circ a_2 \circ a_3) \\ &+ ... \\ &+ \underbrace{\epsilon_{n-1}(a_1 \circ ... \circ a_n))}_{Rundungsfehler}\end{aligned} \tag{3.1.19}$$

Im folgenden Beispiel soll der numerische Rundungsfehler für die einmalige Auswertung der Duffingschen Differentialgleichung abgeschätzt werden.

Beispiel 3.3 für „Rundungsfehleranalyse der Duffingschen Differentialgleichung (Duffing-Oszillator)"
Der ungedämpfte Duffing-Oszillator ist eine nichtlineare Differentialgleichung zweiter Ordnung:

$$\frac{d^2u(t)}{dt^2} = -\omega^2 u(t) + \delta u(t)^3 \tag{3.1.20}$$

mit den Konstanten ω, δ und $u(t)$ als zeitabhängige Variable. Im Folgenden wird eine Rundungsfehleranalyse durchgeführt. Hierbei wird angenommen, dass die Anfangswerte bereits mit einem Eingangsfehler $\epsilon_{u(t)}$ behaftet sind. Ferner sind die verwendeten Konstanten ω, δ nicht mit einem Eingangsfehler behaftet. Um das Ergebnis eines Aufrufs der Kraftfunktion zu ermitteln, sind in diesem Fall 6 Operationen nötig (fünf Multiplikationen und eine Addition), wobei die Reihenfolge genau eingehalten werden muss. Die Gesamtanzahl der Operationen und deren Reihenfolge kann in diesem Fall direkt abgelesen werden:

$$\frac{d^2u(t)}{dt^2} = \underbrace{-\omega\omega u(t)}_{(1)} + \underbrace{\delta u(t)u(t)u(t)}_{(2)} \tag{3.1.21}$$

Hierbei können Teil (1) und (2) unabhängig voneinander berechnet werden. Liegen die Teilergebnisse der beiden Teile vor, dann erst können diese addiert werden. Für das Teilergebnis (1) kann folgende Rundungsfehleranalyse durchgeführt werden: In diesem Fall sind zwei Multiplikationen nacheinander durchzuführen und somit ist $n = 2$. Die Variable $u(t)$ ist mit einem Eingangsfehler behaftet und die Konstante ω nicht. Dadurch ergibt sich folgende Darstellung:

$$\begin{aligned}(1) &:= (\omega \odot \omega) \odot u(t)(1 + \epsilon_{u(t)}) \\ &:= ((\omega \circ \omega)(1 + \epsilon_1)) \odot u(t)(1 + \epsilon_{u(t)}) \\ &:= ((\omega\omega + \omega\omega\epsilon_1))u(t)(1 + \epsilon_2)(1 + \epsilon_{u(t)}) \\ &:= \epsilon_1\epsilon_2\epsilon_{u(t)}u(t)\omega^2 + \epsilon_2\epsilon_u u\omega^2 + \epsilon_1\epsilon_{u(t)}u\omega^2 + \epsilon_{u(t)}u\omega^2 + \epsilon_1\epsilon_2 u\omega^2 + \epsilon_2 u\omega^2 \\ &+ \epsilon_1 u\omega^2 + u\omega^2\end{aligned} \tag{3.1.22}$$

Hier werden Fehlerterme höherer Ordnung vernachlässigt. Das ergibt folgende vereinfachte Formel:

$$(1) := u(t)\omega^2(\epsilon_{u(t)} + \epsilon_2 + \epsilon_1 + 1) \tag{3.1.23}$$

Für das Teilergebnis (2) *kann folgende Rundungsfehleranalyse durchgeführt werden: In diesem Fall sind drei Multiplikationen auszuführen, wobei* δ *nicht und* $u(t)$ *mit einem Eingangsfehler behaftet ist. Der hierbei verwendete Rundungsfehlerindex* ϵ_i *wird aus Teil* (1) *übernommen und fortgeführt, d.h. die erste Multiplikation im Teil* (2) *erhält den Rundungsfehlerindex* $(1+\epsilon_3)$*. Dieser Zusammenhang kann beschrieben werden durch:*

$$\begin{aligned}
(2) &:= \delta \odot u(t)(1+\epsilon_{u(t)}) \odot u(t)(1+\epsilon_{u(t)}) \odot u(t)(1+\epsilon_{u(t)}) \\
&:= \delta u(t)(1+\epsilon_3)(1+\epsilon_{u(t)})u(t)(1+\epsilon_4)(1+\epsilon_{u(t)})u(t)(1+\epsilon_5)(1+\epsilon_{u(t)}) \\
&:= \epsilon_3\,\epsilon_4\,\epsilon_5\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)}^3 + \epsilon_4\,\epsilon_5\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)}^3 + \epsilon_3\,\epsilon_5\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)}^3 + \epsilon_5\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)}^3 \\
&+ \epsilon_3\,\epsilon_4\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)}^3 + \epsilon_4\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)}^3 + \epsilon_3\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)}^3 + \delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)}^3 \\
&+ 3\,\epsilon_3\,\epsilon_4\,\epsilon_5\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)}^2 + 3\,\epsilon_4\,\epsilon_5\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)}^2 + 3\,\epsilon_3\,\epsilon_5\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)}^2 + 3\,\epsilon_5\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)}^2 \\
&+ 3\,\epsilon_3\,\epsilon_4\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)}^2 + 3\,\epsilon_4\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)}^2 + 3\,\epsilon_3\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)}^2 + 3\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)}^2 \\
&+ 3\,\epsilon_3\,\epsilon_4\,\epsilon_5\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)} + 3\,\epsilon_4\,\epsilon_5\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)} + 3\,\epsilon_3\,\epsilon_5\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)} + 3\,\epsilon_5\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)} \\
&+ 3\,\epsilon_3\,\epsilon_4\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)} + 3\,\epsilon_4\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)} + 3\,\epsilon_3\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)} + 3\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)} + \epsilon_3\,\epsilon_4\,\epsilon_5\,\delta\,\mathrm{u}\,(t)^3 \\
&+ \epsilon_4\,\epsilon_5\,\delta\,\mathrm{u}\,(t)^3 + \epsilon_3\,\epsilon_5\,\delta\,\mathrm{u}\,(t)^3 + \epsilon_5\,\delta\,\mathrm{u}\,(t)^3 + \epsilon_3\,\epsilon_4\,\delta\,\mathrm{u}\,(t)^3 + \epsilon_4\,\delta\,\mathrm{u}\,(t)^3 + \epsilon_3\,\delta\,\mathrm{u}\,(t)^3 + \delta\,\mathrm{u}\,(t)^3
\end{aligned} \tag{3.1.24}$$

Das Vernachlässigen der höheren epsilon-Terme erhält man den Ausdruck:

$$(2) := \delta u(t)^3(\epsilon_3 + \epsilon_4 + \epsilon_5 + 3\epsilon_{u(t)} + 1) \tag{3.1.25}$$

Im Anschluss werden die Zwischenergebnisse (1) *und* (2) *noch addiert:*

$$\begin{aligned}
\frac{d^2u(t)}{dt^2} &= -(u(t)\omega^2(\epsilon_{u(t)} + \epsilon_2 + \epsilon_1 + 1)) \odot \delta u(t)^3(\epsilon_3 + \epsilon_4 + \epsilon_5 + 3\epsilon_{u(t)} + 1) \\
&= (u(t)\omega^2(\epsilon_{u(t)} - \epsilon_2 - \epsilon_1 - 1)) + \delta u(t)^3(\epsilon_3 + \epsilon_4 + \epsilon_5 + 3\epsilon_{u(t)} + 1)(1+\epsilon_6) \\
&= 3\,\epsilon_6\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)} + 3\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)} + \omega^2\,\mathrm{u}\,(t)\ \epsilon_{\mathrm{u}(t)} + \epsilon_5\,\epsilon_6\,\delta\,\mathrm{u}\,(t)^3 + \epsilon_4\,\epsilon_6\,\delta\,\mathrm{u}\,(t)^3 \\
&+ \epsilon_3\,\epsilon_6\,\delta\,\mathrm{u}\,(t)^3 + \epsilon_6\,\delta\,\mathrm{u}\,(t)^3 + \epsilon_5\,\delta\,\mathrm{u}\,(t)^3 + \epsilon_4\,\delta\,\mathrm{u}\,(t)^3 + \epsilon_3\,\delta\,\mathrm{u}\,(t)^3 + \delta\,\mathrm{u}\,(t)^3 \\
&- \epsilon_2\,\omega^2\,\mathrm{u}\,(t) - \epsilon_1\,\omega^2\,\mathrm{u}\,(t) - \omega^2\,\mathrm{u}\,(t)
\end{aligned} \tag{3.1.26}$$

Das Vernachlässigen der Terme mit höherer Ordnung ergibt:

$$\begin{aligned}
\frac{d^2u(t)}{dt^2} &= 3\,\epsilon_6\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)} + 3\,\delta\,\mathrm{u}\,(t)^3\,\epsilon_{\mathrm{u}(t)} + \omega^2\,\mathrm{u}\,(t)\ \epsilon_{\mathrm{u}(t)} + \epsilon_5\,\delta\,\mathrm{u}\,(t)^3 + \epsilon_4\,\delta\,\mathrm{u}\,(t)^3 \\
&+ \epsilon_3\,\delta\,\mathrm{u}\,(t)^3 - \epsilon_2\,\omega^2\,\mathrm{u}\,(t) - \epsilon_1\,\omega^2\,\mathrm{u}\,(t) + \delta\,\mathrm{u}\,(t)^3 - \omega^2\,\mathrm{u}\,(t) \\
&= \underbrace{-\omega^2 u(t) + \delta\,\mathrm{u}\,(t)^3}_{I} - \underbrace{\omega^2 u(t)(\epsilon_1 + \epsilon_2 - \epsilon_{\mathrm{u}(t)})}_{II} + \underbrace{\delta u(t)^3(\epsilon_3 + \epsilon_4 + \epsilon_5 + 3\epsilon_{\mathrm{u}(t)} + 3\epsilon_6\,\epsilon_{\mathrm{u}(t)})}_{III}
\end{aligned}$$

Dieser Ausdruck kann in drei Hauptanteile zerlegt werden. Der Teil I *beschreibt die Gleichung des Duffing Oszillators. Die Teile* II *und* III *enthalten die Fehleranteile mit unterschiedlichen Verstärkungsfaktoren. Teil* II *wird einfach durch den Wert von* $u(t)$ *verstärkt, wohingegen der dritte Teil (*III*) mit der dritten Potenz von* $u(t)$ *verstärkt wird. Im Falle von* $\delta = 1$ *und* $\omega = 1$ *muss zwischen den Fällen unterschieden werden:*

1. $u(t) < 1$*:*
   Es dominiert der betragsmäßige Fehleranteil II *und verursacht somit den größten Rundungsfehleranteil.*

2. $u(t) > 1$:
   Es dominiert der Term III und liefert den größten Fehleranteil. Zusätzlich führt der Eingangsfehler zu einer Verstärkung um den Faktor 3.

Wie dieses Beispiel gezeigt hat, wächst die Anzahl der Ausdrücke schon bei einfachen Formeln sehr schnell an. Das beschriebene Vorgehen zur Rundungsfehleranalyse ist somit nur für kleine Ausdrücke mit weniger als 10 Verknüpfungen geeignet. Um für umfangreiche Gleichungssysteme bzw. Algorithmen den Rundungsfehler und dessen Verstärkung formal zu erfassen, wird der Einsatz von Computeralgebrasystemen wie MATHEMATICA[12], MAXIMA[13] oder Axiom[14] empfohlen. Ferner kann auch SymbolicC++[15] oder GiNAC[16] dafür verwendet werden.
Darüber hinaus muss noch angemerkt werden, dass die Rundungsfehleranalyse für Winkelfunktionen sehr schwer durchführbar ist. Diese Funktionen sind zwar Bestandteil der Programmiersprache, ihre interne Implementierung ist jedoch plattformabhängig und unterscheidet sich infolgedessen in vielen Fällen grundlegend voneinander.
Um im Allgemeinen die Analyse von Ausdrücken, basierend auf den vier Grundrechenarten zu erleichtern, ist Tabelle 3.5 zu beachten. Dort wurde für einige in der Mathematik häufig verwendete Ausdrücke die Rundungsfehleranalyse durchgeführt. Dies soll als Ausgangspunkt für umfangreichere Analysen dienen. Die in Tabelle 3.5 verwendeten Variablen a, b, c, d, e, f sind nicht durch Eingangsfehler verfälscht.

Ausdruck	Rundungsfehler
$a^2 \oplus b^2$	$a^2 + b^2 + \epsilon_1 a^2 + \epsilon_2 b^2 + \epsilon_3(a^2 + b^2)$
$a^2 \oplus b^2 \oplus c^2$	$a^2 + b^2 + c^2 + \epsilon_1 a^2 + \epsilon_2 b^2 + \epsilon_3(a^2 + b^2) + \epsilon_4 c^2 + \epsilon_5(a^2 + b^2 + c^2)$
$\frac{a}{b} \oplus \frac{c}{d}$	$\frac{a}{b} + \frac{c}{d} + \epsilon_1 \frac{a}{b} + \epsilon_2 \frac{c}{d} + \epsilon_3(\frac{a}{b} + \frac{c}{d})$
$\frac{a}{b} \oplus \frac{c}{d} \oplus \frac{e}{f}$	$\frac{a}{b} + \frac{c}{d} + \frac{e}{f} + \epsilon_1 \frac{a}{b} + \epsilon_2 \frac{c}{d} + \epsilon_3(\frac{a}{b} + \frac{c}{d}) + \epsilon_4 \frac{e}{f} + \epsilon_5(\frac{a}{b} + \frac{c}{d} + \frac{e}{f})$

Tabelle 3.5: Auflistung häufig vorkommender Ausdrücke

Die Rundungsfehleranalyse ist ein weites Feld und somit als eigene Disziplin zu betrachten. Die hier aufgeführten Beispiele sollen eine Hilfestellung bieten, um an numerisch *sensiblen* Programmstellen die Stabilität eines Algorithmus überprüfen zu können. Ferner kann hiermit speziell bei iterativen Verfahren der entstehende Rundungsfehler für jede Iteration angegeben werden und dadurch eine gute Fehlerabschätzung durchgeführt werden. Außerdem wird in [Jordan72] (ab Seite 291) am Beispiel des Euler-Cauchy Integrationsverfahrens eine detaillierte Fehlerabschätzung durchgeführt, welche auch den entstehenden Rundungsfehler berücksichtigt. Ferner wird auf [Burlisch05] (ab Seite 128) hingewiesen, wo der Einfluss des Rundungsfehlers auf Einschrittverfahren zur Lösung gewöhnlicher Differentialgleichung abgeschätzt wird. Dort wird angemerkt, dass der unvermeidbare Rundungsfehler das Ergebnis erst ab einer bestimmten Integrationsschrittweite verschlechtert.

[12] siehe dazu [Mathematica]
[13] siehe dazu [Maxima]
[14] siehe dazu [Axiom]
[15] siehe dazu [Hardy08]
[16] siehe dazu [GiNAC]

3.2 Erweiterung der Stellengenauigkeit

Um mit höherer Genauigkeit als im IEEE 754 - Standard festgelegt, rechnen zu können ist es erforderlich, die Anzahl der Mantissenbits p zu vergrößern. Um andererseits den verfügbaren darstellbaren Zahlenbereich zu erweitern, muss die Zahl der Exponentenbits r inkrementiert werden. Dieser Weg wurde bei den in Abschnitt 4 (ab Seite.87)

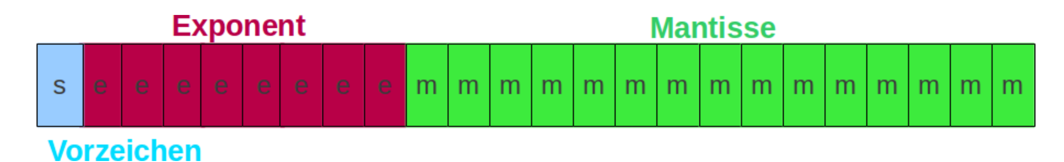


Abbildung 3.1: Interne Darstellung einer Gleitkommazahl für r = 8 und p = 15

vorgestellten numerischen Bibliotheken beschritten. Dadurch ist es möglich, die Genauigkeit von Rechenoperationen beliebig einzustellen. Die einzustellende Genauigkeit wird lediglich durch den limitiert vorhanden Speicher begrenzt.

3.3 Rechnung mit Festpunktzahlen

Der Begriff der Festpunktzahlen soll hier nur aus Gründen der Vollständigkeit erwähnt werden. Diese Zahlen besitzen eine feste Anzahl an Ziffern, wobei die Position des Dezimalpunkts festgelegt ist. Anders als bei Gleitkommazahlen, welche i.d.R. normalisiert dargestellt werden, fällt in dieser Darstellungsform die Normalisierung weg. Aufgrund der festen Position des Dezimalpunkts können die meisten Operationen durch einfache Schiebeoperationen realisiert werden, was in vielen Fällen erhebliche Vorteile in der Abarbeitungsgeschwindigkeit bringt. Leider existieren aktuell[17] keine frei verfügbaren *multiprecision*-Bibliotheken auf Festpunktbasis mit angemessenem Funktionsumfang.

3.4 Auswahl einer geeigneten Programmiersprache und Laufzeitumgebung

Für die Auswahl einer geeigneten Programmiersprache ist es erforderlich, die Aufgabenstellung dieser Arbeit und deren Randbedingungen genauer zu betrachten. Ziel ist es, ein Werkzeug zur Lösung gewöhnlicher Differentialgleichungen mit frei wählbarer Anzahl signifikanter Nachkommastellen zu schaffen. Zur Lösung gewöhnlicher Differentialgleichungen existiert eine Vielzahl verschiedener Algorithmen, welche prinzipiell mit einer beliebigen Programmiersprache realisiert werden könnten. Die zusätzliche Anforderung der frei wählbaren Anzahl signifikanter Nachkommastellen schränkt die Auswahl möglicher Programmiersprachen auf jene ein, die Datentypen[18] zur genauen Rechnung bereitstellen. Außerdem soll das hier geschaffene Werkzeug mit einer

[17] Stand Oktober 2011

[18] Datentypen oder Bibliotheken zur Rechnung mit frei wählbarer Anzahl signifikanter Nachkommastellen

bewährten und - im Sinne der Laufzeit - effizienten Programmiersprache implementiert werden, um die Verbreitung und Nachhaltigkeit des Werkzeugs zu erhöhen. Als Entscheidungshilfe können die Langzeitanalysen[19] der meist genutzten Programmiersprachen helfen. In Abbildung 3.2 ist diese Verteilung für den Zeitraum von 2002 bis 2012 aufgetragen[20]. Daraus wird ersichtlich, dass objektorientierte Sprachen wie C++,

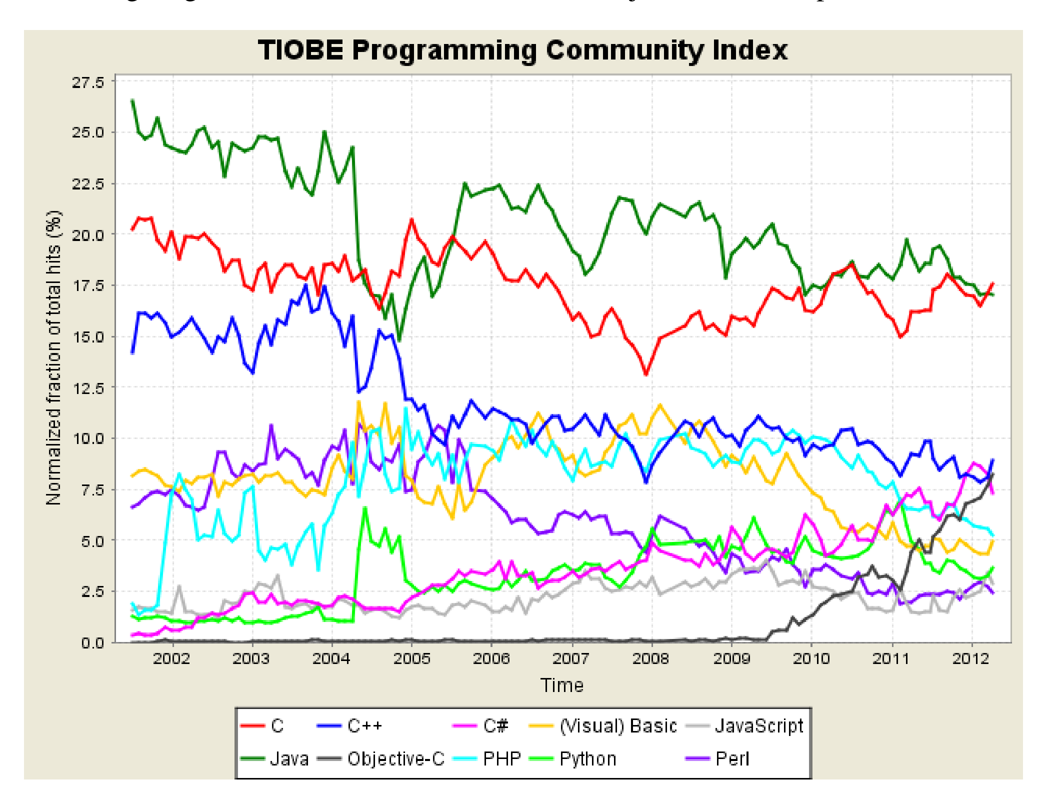


Abbildung 3.2: Langzeitanalyse des Anteils der zehn meist genutzten Programmiersprachen laut TIOBE-Index

Java oder Objective-C weit verbreitet sind. Außerdem kann abgelesen werden, dass die Sprache Objective-C erst seit Kurzem häufig genutzt wird. Auch prozedurale Sprachen, wie beispielsweise die Programmiersprache C, sind noch weit verbreitet[21]. Wird alleinig die Verteilung in Betracht gezogen, so fällt die Wahl der Programmiersprache auf C++. Diese ist objektorientiert (siehe Abschnitt 3.5), besitzt die nötige Verbreitung, um die geforderte Nachhaltigkeit zu gewährleisten und wird aktiv weiterentwickelt[22]. Außerdem inkludiert die Programmiersprache C++ die prozedurale C, wodurch sich bei der Implementierung auf dedizierten Hardwareplattformen[23] Vorteile hinsichtlich der Programmlaufzeit erzielen lassen. Darüber hinaus bietet C++ die Möglichkeit der

[19] Siehe mehr dazu unter http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html, Download am 11.04.2012

[20] Diese Grafik wurde der Webseite http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html entnommen (Download am 11.04.2012).

[21] Große Projekte wie der Linux-Kernel oder der GNU-Compiler sind in der Programmiersprache C implementiert.

[22] Informationen zum neuen Sprachstandard können auf der Webseite http://www.open-std.org/jtc1/sc22/wg21/ (Download am 11.04.2012) nachgeschlagen werden.

[23] Siehe dazu auch Abschnitt 5.2.

Templateprogrammierung, mit der redundante Implementierungen gezielt vermieden werden können. Außerdem hat die Sprache C++, verglichen mit prozeduralen Sprachen (z.B. FORTRAN), einen syntaktisch strukturierten Aufbau. Dies ermöglicht eine nachträgliche Konvertierung in andere objektorientierte Sprachen und ist somit zukunftsweisend. Als Laufzeitumgebung wurde das freie Betriebssystem Linux gewählt, da es ohne eventuell anfallende Lizenzgebühren zur Verfügung steht.

3.5 Konzepte objektorientierter Programmierung

In den folgenden Unterpunkten wird kurz auf die grundlegenden Konzepte der objektorientierten Programmierung eingegangen. Darüber hinaus werden die Paradigmen der Kapselung, Vererbung, Operatorüberladung und die C++-Programmierung mit Templates erklärt.

3.5.1 Objektorientierte Programmierung

[OOP] Objektorientierte Programmierung

Die Wurzeln der Sprache C++ liegen in der Programmiersprache C. C++ ist syntaktisch und semantisch eine Obermenge der Programmiersprache C. Die Programmiersprache C++ verfolgt das Konzept der OOP. Dadurch wird die Flexibilität und die Wiederverwendbarkeit von Programmen gefördert. Das Grundkonzept der OOP ist, Daten und Funktionen möglichst eng in einem sogenannten Objekt zusammenzufassen und nach außen hin zu kapseln. Dies hilft dedizierte Zugangspunkte zu den gekapselten Daten zu schaffen. Die jeweiligen Zugangspunkte werden über Funktionen realisiert, welche den Klassen zugeordnet sind. Im Gegensatz dazu beschrieb das früher vorherrschende Programmierparadigma (*prozedurale Programmierung*) eine Unterteilung zwischen Funktionen und Daten[24].

Begriffsdefinition DEF3.1 „Klasse und Objekt“:

Eine Klasse stellt im Sinne der OOP eine Datenstruktur dar, welche Funktionen und Daten in sich vereint. Diese Datenstruktur wird in Form einer Klassendefinition festgelegt. Ein Objekt stellt eine konkrete Realisierung einer Klasse dar. Mit Hilfe der enthaltenen Funktionen kann der Zustand des Objekts auf eine definierte Art und Weise beeinflusst werden. Eine besondere Funktion, die jede Klasse besitzt, ist der Konstruktor. Dieser stellt den Einsprungspunkt beim Erzeugen des Objekts dar. Innerhalb des Konstruktors wird i.d.R. ein Objekt in einen definierten Zustand versetzt, in dem die Variablen der Klassen mit Standardwerten initialisiert werden.

Begriffsdefinition DEF3.2 „Kapselung“:

Der Begriff der Kapselung wird in der Programmierung mit dem gezielten Verbergen von Implementierungsdetails assoziiert. Der direkte Zugriff auf die interne Datenstruktur wird dabei verhindert. Der Zugriff erfolgt hier über eigens definierte Schnittstellen. Laut Definition stellt ein Objekt eine in sich abgeschlossene Einheit, ähnlich einer black box dar. Von außen betrachtet stehen dem Programmierer lediglich die Schnittstellen zur Verfügung, die interne Darstellung bzw. die Implementierung bleibt ihm dabei verborgen. Durch diese Abstraktion ist es möglich, komplexe Implementierungsdetails zu verbergen. Dadurch können Objekte als in sich abgeschlossene Einheiten angesehen werden, welche ein definiertes Verhalten besitzen.

[24] siehe dazu [Auperle02]

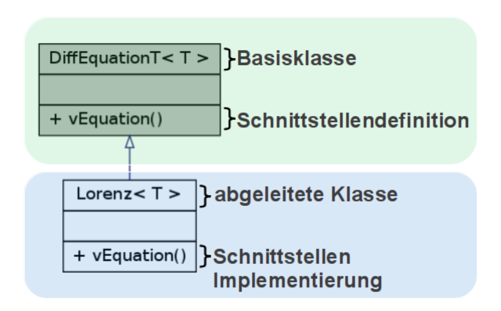


Abbildung 3.3: Vererbung von Schnittstellenfunktionen an eine abgeleitete Klasse

Dieses Konzept wurde bei der Implementierung der Software zur numerischen Integration eingesetzt. Dabei wurde jedes der implementierten Verfahren in eine eigene Klasse gekapselt. Somit besitzt die Klasse ausschließlich definierte Schnittstellen und die interne Funktionalität ist von außen nicht einsehbar.

Begriffsdefinition DEF3.3 „Vererbungsprinzip“:

Das Konzept der Vererbung ermöglicht die Wiederverwendung vorhandener Klassendefinitionen. Hieraus können wieder neue Klassen definiert werden, indem deren Funktionalität vererbt *und verfeinert wird. Dadurch ist es beispielsweise möglich, die Funktionalität einer Basisklasse in die eigene Klassendefinition mit aufzunehmen und einige zusätzliche Aspekte hinzuzufügen, ohne die Basisklasse zu verändern. Die Vererbung kann somit auch als eine Art der Spezialisierung angesehen werden.*

Dieses Konzept wurde beispielsweise bei der Definition einer Klasse eingesetzt, welche die Implementierung einer Differentialgleichung enthält. Dabei wurde erst eine Basisklasse definiert, welche die nötigen Schnittstellenfunktionen zum Aufruf der rechten Seite einer Differentialgleichung enthalten soll. Diese Klasse besitzt zunächst noch keine konkrete Differentialgleichung, sondern definiert lediglich die Schnittstellenfunktion in allgemeiner Form. Konkretisiert wird dieses Konzept erst, wenn eine Klasse von dieser Basisklasse abgeleitet wird und damit die Schnittstellen der Basisklasse erbt. Folglich wird auch die Gestalt der Funktionsdefinition, die zum Aufruf der entsprechenden Funktion nötig ist, auf die abgeleitete Klasse übertragen. Durch Überschreiben dieser speziellen Schnittstellenfunktion innerhalb der abgeleiteten Klasse erfolgt eine Spezialisierung, d.h. es wird eine konkrete Differentialgleichung hinterlegt, wie z.B. die Differentialgleichung des Lorenz-Attraktors. Dies hat den Vorteil, dass alle Algorithmen zur Lösung von Differentialgleichungen lediglich die Schnittstelle der Basisklasse kennen und dadurch eine Trennung zwischen konkreter Aufgabenstellung (z.B. Differentialgleichung) und numerischem Lösungsverfahren gewahrt bleibt. In Abbildung 3.3 wird anhand eines konkreten Beispiels die Ableitungsbeziehung in UML[25] konformer Notation dargestellt. Darin leitet die Klasse *Lorenz* von der Klasse *DiffEquationT* ab. Diese wird somit zur Basisklasse der Klasse *Lorenz*, welche die Schnittstellendefinition erbt. Durch einfaches Implementieren (Hinterlegen einer konkreten Differentialgleichung) innerhalb der Klasse *Lorenz* wird eine standardisierte Schnittstellenkonvention aufrecht erhalten. In Listing 3.1 ist dieser Zusammenhang als C++-Quelltext ausgedrückt. Dies ist eine reduzierte Form der konkreten Imple-

[UML] Unified Modelling Language

[25]siehe dazu [Erler00]

mentierung und soll lediglich das Prinzip der Vererbung in den Vordergrund rücken, da ein Großteil der hier entwickelten Software auf diesem Prinzip aufbaut. In Zeile 4 wird die Funktion *vEquation* definiert, jedoch nicht implementiert, und in Zeile 9 erfolgt die Implementierung. An dieser Stelle wird die jeweilige Differentialgleichung definiert.

```
1 template <typename T> class DiffEquationT
2 {
3     public:
4             virtual void vEquation() = 0;
5 };
6 template <typename T> class Lorenz : public DiffEquationT<T>
7 {
8     public:
9             void vEquation(){}
10 };
```

Listing 3.1: Quelltextauszug zur Vererbung von Schnittstellenfunktionen

Noch hat dieses Prinzip keinen Vorteil, da eine konkrete Anwendung für die standardisierte Schnittstelle fehlt. Zur Lösung von Differentialgleichungen werden Integratoren verwendet, welche genau diese Standardschnittstelle benötigen. Es ist außerdem vorteilhaft diese Algorithmen nur einmal zu implementieren und die jeweilige Aufgabenstellung (Differentialgleichung) dem Verfahren zu einem späteren Zeitpunkt mitzuteilen. In Listing 3.2 ist ein Beispiel abgebildet, worin die Verwendung einer abstrakten Schnittstellendefinition vorteilhaft ist. Darin ist eine Klasse *IntegratorT* definiert. Diese besitzt einen Memberpointer vom Typ der abstrakten Basisklasse *DiffEquationT*. Diese wird während des Aufrufs im Konstruktor initialisiert (Zeile 5). In Zeile 21 wird nun ein Objekt einer konkrete Differentialgleichung erzeugt und in Zeile 22 dem Konstruktor der Klasse *IntegratorT* übergeben. Schließlich kann in der Funktion *vStep* in der Klasse *IntegratorT* die Funktion *vEquation* aufgerufen werden (siehe Zeile 11 und 23). Somit ist die Implementierung des Algorithmus und die Implementierung der konkreten Aufgabenstellung getrennt und kann nach Bedarf miteinander verknüpft werden.

```
1 template <typename T> class IntegratorT
2 {
3     public:
4         explicit IntegratorT(DiffEquationT<T> *DiffEquationPtr)
5         :m_pDiffEquation(DiffEquationPtr)
6         {
7         }
8
9         void vStep()
10        {
11            m_pDiffEquation->vEquation();
12        }
13
14    private:
15
16        DiffEquationT<T> *m_pDiffEquation;
17 };
18
19 int main()
```

```
20 {
21     Lorenz<double> LorenzEquation;
22     IntegratorT<double> Integrator(&LorenzEquation);
23     Integrator.vStep();
24     return 0;
25 }
```

Listing 3.2: Quelltextauszug zur Nutzung von Schnittstellendefinitionen

Begriffsdefinition DEF3.4 „Operatorüberladung“:
Die Operatorüberladung ist ein überaus nützliches Werkzeug der OOP. Damit ist es beispielsweise möglich, die Rechenoperatoren wie $+, -, *, /$ mit neuen Funktionen auszustatten bzw. diese Operanden für bestimmte Zwecke neu zu definieren. Für alle primitiven Standarddatentypen sind diese Überladungen bereits vorhanden. Dadurch wird die Addition von allen primitiven numerischen Datentypen ermöglicht. Das Konzept erlaubt es, die für jede Klasse notwendigen Operatoren zu überladen.

Das Prinzip der Operatorüberladung wurde in dieser Arbeit massiv eingesetzt. Beispielweise wurde eine Klasse (*VectorT*) zur Speicherung von Daten in Form eines Vektors definiert und dort alle nötige Operatoren überladen.

Begriffsdefinition DEF3.5 „Templateprogrammierung“:
Das Konzept der Templates ermöglicht es, eine vom Datentyp unabhängige Programmierung zu realisieren. Dabei wird der verwendete Datentyp variabel definiert. Wird dieses Konzept durchgehend umgesetzt, dann ist zur Implementierung eines bestimmten Algorithmus nur eine einzige Templateklasse nötig. Diese besondere Art der Unabhängigkeit vom verwendeten Datentyp fördert die Wiederverwendbarkeit im Sinne der Softwareentwicklung.

Alle Funktions- und Klassendefinition, welche im Rahmen dieser Arbeit entwickelt wurden, sind als Templates realisiert. Ansonsten hätte jedes der Lösungsverfahren eine eigene, auf den verwendeten Datentyp zugeschnittene Implementierung bekommen müssen. Folglich braucht nur ein Quelltext gepflegt zu werden und es werden Mehrdeutigkeiten unterschiedlicher Realisierungen eines bestimmten Verfahrens implizit ausgeschlossen. Im Allgemeinen wird eine Templatefunktion in der Programmiersprache C++ wie folgt formuliert.

```
1 template <typename T> void Funktionsname(Parameter)
2 {Funktionsrumpf}
```

Listing 3.3: Definition einer Templatefunktion

Durch die syntaktische Deklaration `template <typename T>` wird der Datentyp durch den Schablonenparameter `T` eingeführt. Dieser kann im Funktionsrumpf wie ein herkömmlicher Datentyp verwendet werden. Die Aufrufe dieser Templatefunktion für die Datentypen *bigfloat* und *double* lauten dann wie folgt:

```
1 Funktionsname<bigfloat>();
2 Funktionsname<double>();
```

Listing 3.4: Instanziierung einer Templatefunktion

Wird dieser Quelltext übersetzt, so erfolgt in der Präprozessierungsphase des Compilers eine Instanziierung der Templates, d.h. erst zu diesem Zeitpunkt wird aus einer Templatefunktion eine herkömmliche Klasse im Sinne der OOP.

3.5.2 Vektor - und Matrixoperationen

Zur Rechnung mit Vektoren und Matrizen wurden die Klassen `VectorT` und `MatrixT` als Templateklassen implementiert. Durch die bereits vorgestellte Operatorüberladung enthalten diese Klassen einen Großteil der Grundoperationen, welche zur Rechnung mit Matrizen und Vektoren erforderlich sind.

3.6 Objektorientierte Implementierung der Integrationsverfahren

Die verschiedenen Algorithmen zur Lösung gewöhnlicher Differentialgleichungen wurden jeweils in eine eigene Klasse gekapselt. Aus der Sicht eines Anwenders sind die unterschiedlichen Integratoren Werkzeuge zur Lösung gewöhnlicher Differentialgleichungen. Somit soll jeder der Algorithmen möglichst ohne Abhängigkeiten einsetzbar sein. Dies wird zunächst realisiert, indem jedes der implementierten Lösungsverfahren von einer Basisklasse (hier *IntegratorBaseT*) ableitet. Infolgedessen erzwingt dies eine Standardisierung der Schnittstelle eines Integrationsverfahrens im Allgemeinen. In Abb. 3.4 ist dieser Zusammenhang exemplarisch dargestellt. Besonderes Augenmerk

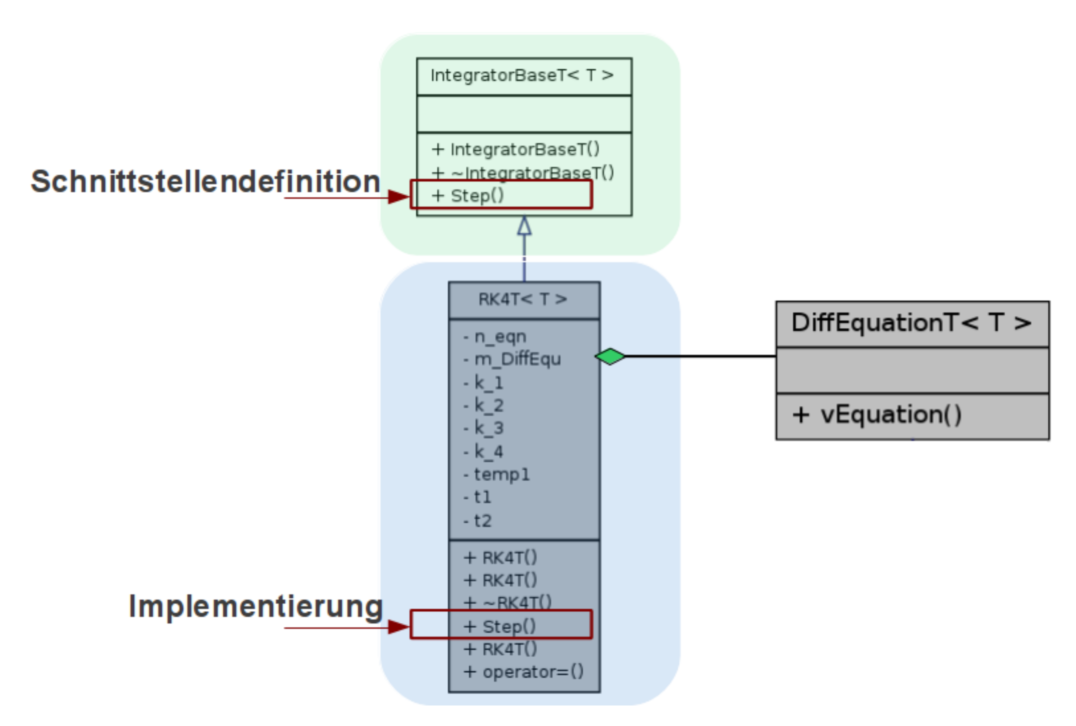


Abbildung 3.4: Klassendiagramm der Klasse RK4 (Runge-Kutta 4.Ordnung)

soll hier auf die Funktion *vStep* gelegt werden. Deren Schnittstelle wird in der Basisklasse *IntegratorBaseT* definiert und in der abgeleiteten Klasse *RK4T* implementiert.

Außerdem besitzt die Klasse *RK4T* eine Zeigervariable der abstrakten Basisklasse *DiffEquationT*. Diese wird benötigt, um eine bestimmte Aufgabenstellung (Differentialgleichungen) zu lösen (siehe mehr dazu im Abschnitt 3.5.1). Im Folgenden soll die Verwendung dieses Verfahrens in einem kurzen Quelltextauszug gezeigt werden:

```
1  // define datatype
2  typedef double T;
3  // create local variables
4  T Starttime=0.0,Stepsize=0.1;
5  // create an vector of dimension = 2
6  VectorT<T> StartVector(2);
7             StartVector[0]=1;
8             StartVector[1]=0;
9  // create a DiffEquation object
10 DuffingT<T> Duffing;
11 // create a Runge-Kutta object
12 RK4T<T> RK4(&Duffing, StartVector.size());
13 // perform one integration step
14 RK4.vStep(Starttime, StartVector, Stepsize);
```

Listing 3.5: Quelltextauszug zur Verwendung eines speziellen Integrationsverfahrens

In Listing 3.5 wird erst eine Verallgemeinerung des verwendeten Datentyps (hier *double*, Zeile 2) vorgenommen. Anschließend wird die Startzeit und die Schrittweite definiert (Zeile 4). In Zeile 6 wird ein *VectorT*-Objekt erzeugt und in den darauf folgenden Zeilen werden die Anfangswerte zur Integration gespeichert. Des Weiteren wird in Zeile 10 ein Objekt der *DuffingT*-Klasse angelegt. Darin ist die Differentialgleichung des ungedämpften Duffing-Oszillators hinterlegt. Anschließend wird in Zeile 12 eine Instanz des Runge-Kutta-Integrationsverfahrens erzeugt. Der Konstruktor der *RK4T*-Klasse verlangt einen Zeiger auf ein Objekt vom Typ *DiffEquationT* und die Dimension des Startvektors in der Parameterliste. Schließlich wird ein Integrationsschritt durchgeführt, indem die Funktion *vStep* aufgerufen wird. Nach dem Aufruf enthält der *Startvector* die Werte zum Zeitpunkt $t_{start} + h$, wobei h ein Platzhalter für die Schrittweite ist.

3.6.1 Kapselung der Integrationsverfahren in der Klasse *IntegratorT*

Im Abschnitt 3.6 wurde auf die objektorientierte Implementierung eines speziellen Integrationsverfahrens verwiesen. Da im Rahmen dieser Arbeit unterschiedliche Integrationsverfahren umgesetzt wurden, ist es nötig, diese in einer Klasse (*IntegratorT*) zusammenzufassen. Die hierzu verwendeten Vererbungsbeziehungen sind in Abb. 3.5 dargestellt. Im Sinne der Softwareentwicklung wurde an dieser Stellte das *Fassade*-Entwurfsmuster[26] umgesetzt. Im Allgemeinen bietet es eine einheitliche Schnittstelle zu einem Subsystem verschiedener Klassen. In diesem Fall wird ein zentraler Zugangspunkt zu den jeweils implementierten Integrationsverfahren geschaffen. Wird die *IntegratorT*-Klasse verwendet, so kann das verwendete Lösungsverfahren lediglich durch Abwandlung eines Attributes geändert werden, was die Implementierung erleichtert. Dies soll durch Listing 3.6 veranschaulicht werden. In den Zeilen eins bis zehn werden analog zum vorangegangenen Beispiel die Startwerte initialisiert und ein Objekt

[26]Mehr Informationen zu diesem und anderen Entwurfsmustern der Softwareentwicklung findet man bei [Gamma01].

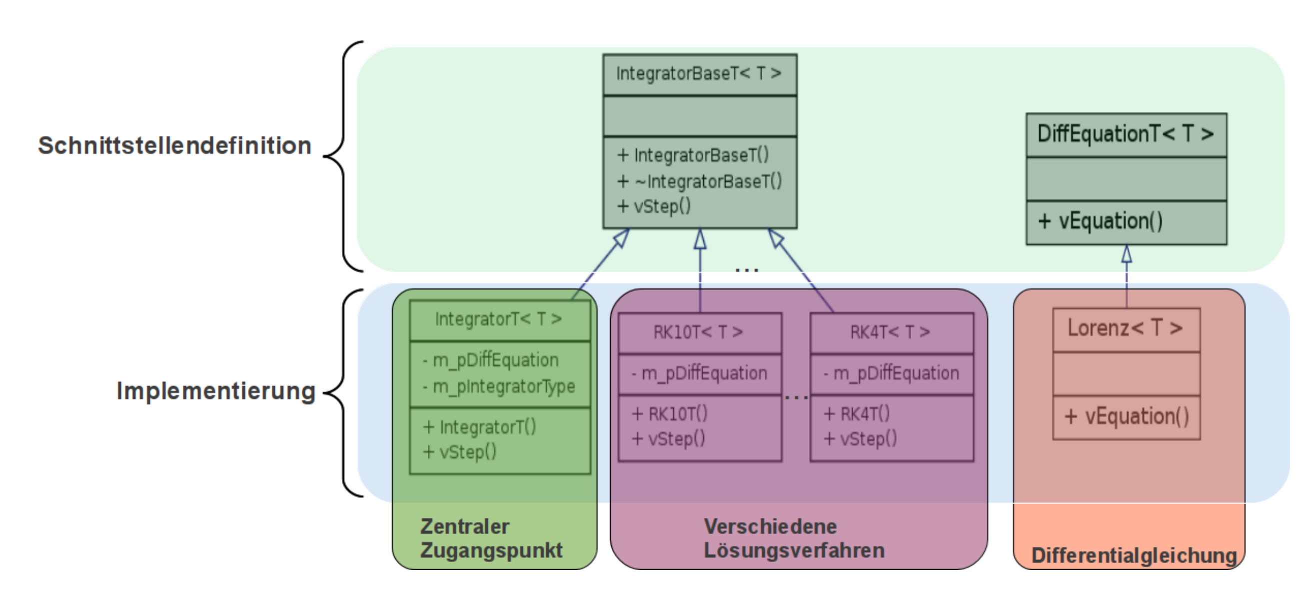


Abbildung 3.5: IntegratorT als zentraler Zugangspunkt zur Integration von Differentialgleichungen

zur Lösung einer gewöhnlichen Differentialgleichung angelegt. In Zeile zwölf wird ein Objekt vom Type *IntegratorT* erzeugt. Zur Erzeugung eines *IntegratorT*-Objekts werden identische Parameter, analog zu den Integrationsverfahren, benötigt. Daraufhin wird in Zeile 14 das zur Berechnung verwendete Integrationsverfahren festgelegt. In diesem Fall wird das Runge-Kutta-Verfahren (4.Ordnung) ausgewählt. Anschließend wird in Zeile 16 ein Integrationsschritt durchgeführt. Um das Umschalten zwischen den verschiedenen Lösungsverfahren zu demonstrieren, wird in Zeile 18 auf das Runge-Kutta-Verfahren (10.Ordnung) umgeschaltet.

```
1  // define datatype
2  typedef double T;
3  // create local variables
4  T Starttime=0.0,Stepsize=0.1;
5  // create an vector of dimension = 2
6  VectorT<T> StartVector(2);
7             StartVector[0]=1;
8             StartVector[1]=0;
9  // create a DiffEquation object
10 LorenzT<T> Lorenz;
11 // create an IntegratorT object
12 IntegratorT<T> Integrator(&Lorenz, StartVector.size());
13 // set integrator type
14 Integrator.vSetIntegratorType(RK4);
15 // perform one integration step
16 Integrator.vStep(Starttime, StartVector, Stepsize);
17 // change integrator type to RK10
18 Integrator.vSetIntegratorType(RK10);
```

Listing 3.6: Quelltextauszug zur Verwendung der *IntegratorT*-Klasse

Kapitel 4

Eignung von *multiprecision*-Bibliotheken

Schwerpunkt des Kapitels:

Aufgrund der Vielzahl frei verfügbarer **multiprecision**-Bibliotheken für die Programmiersprache C/C++, ist es nötig, eine Evaluierung hinsichtlich funktionaler Ausstattung und deren Laufzeitverhalten durchzuführen. Folgende Bibliotheken wurden untersucht:

- LiDIA
- NTL
- CLN
- MAPM
- GMP

4.1 Auswahl einer geeigneten *multiprecision*-Bibliothek

Neben kryptographischen[1] und zahlentheoretischen Anwendungen ist die Rechnung mit höherer Anzahl signifikanter Nachkommastellen für die Lösung von Bewegungsproblemen ein außerordentlich wertvolles Hilfsmittel. Damit ist es möglich, die gerade verwendeten Algorithmen auf ihre numerische Stabilität und Robustheit zu testen. Soll eine Simulation mit höherer Genauigkeit durchgeführt werden, so dauert dies, verglichen mit herkömmlich verfügbarer Genauigkeit, in der Regel länger. Ein Grund dafür ist, dass die Berechnung einer Zahl mit höherer Genauigkeit in Software abgebildet werden muss, ganz im Gegensatz zu den Basisdatentypen, welche i.d.R. direkt auf der CPU verarbeitet werden. Aus diesem Grund ist es entscheidend zu wissen, welche der zahlreich verfügbaren Bibliotheken zur Rechnung mit frei wählbaren Stellengenauigkeiten für die aktuelle Aufgabenstellung am geeignetsten ist. Zudem gibt es, allein für die Programmiersprachen C/C++, eine Vielzahl verschiedener *open-source*-Bibliotheken. Eine kurze Übersicht zu einigen aktuell[2] verfügbaren Bibliotheken, kategorisiert nach den jeweils mitgelieferten Datentypen, ist in Tabelle 4.1 angegeben[3].

[CPU] Central Processing Unit

Name	Version	Integer	Float	Decimal	Complex	Rational
LiDIA	2.3.0	✓	✓	-	✓	✓
NTL	5.5.2	✓	✓	-	-	-
CLN	1.3.2	✓	✓	-	✓	✓
GMP	5.0.2	✓	✓	-	-	✓
ttmath	0.9.2	✓	✓	-	-	-
MAPM	4.9.5a	✓	-	✓	-	-

Tabelle 4.1: Übersicht zu *multiprecision*-Bibliotheken für die Programmiersprache C++

Prinzipiell können die jeweiligen Auswahlkriterien in zwei Hauptkategorien eingeteilt werden:

- **Funktionsumfang**
  Der unterstützte Funktionsumfang, hinsichtlich mathematischer Transformationen, ist eines der wichtigsten Kriterien bei der Bewertung von *multiprecision*-Bibliotheken. Ohne diesen grundlegenden Funktionsumfang ist die Bibliothek bzw. der jeweilig enthaltene *multiprecision*-Datentyp nur begrenzt nutzbar.

- **Abarbeitungsgeschwindigkeit**
  Als ebenfalls wichtiges Kriterium kann die Abarbeitungsgeschwindigkeit angesehen werden. Diese kann sich, abhängig von der untersuchten Problemstellung und Hardwareplattform, unterschiedlich verhalten. Somit kann keine pauschale Antwort gegeben werden, welcher Datentyp die beste Abarbeitungsgeschwindigkeit besitzt. Dennoch können synthetische Leistungsmessungen (*Benchmarks*)

[1] siehe mehr dazu unter [Denis06]

[2] November 2011

[3] Es gibt noch weitere Bibliotheken mit ähnlicher Ausstattung an Datentypen. Diese werden hier bewusst nicht erwähnt, da eine vollständige Gegenüberstellung den Rahmen dieser Arbeit sprengen würde.

eine gute Hilfestellung bei der Auswahl bieten. Die letztendlich erzielte Abarbeitungsgeschwindigkeit kann weiter nach Einflußgrößen aus Hard - und Software kategorisiert werden:

– Hardware:
  * Welche Hardwarearchitektur liegt zu Grunde (32/64-Bit CPU)?
  * Mit welcher Frequenz ist die CPU getaktet?
  * Wie viele Rechenkerne besitzt die CPU?
  * Welche Menge RAM steht auf der Zielplattform zur Verfügung?
– Software:
  * Wurde der Test auf einem Echtzeitbetriebssystem durchgeführt?
  * Welcher Compiler wurde zum Übersetzen der Anwendung verwendet?
  * Welche Optimierungen wurden durch den Compiler vorgenommen?
  * Wurde der Quellcode handoptimiert, evtl. durch *inline*-Assembler?
  * Welche arithmetische Operation wird am häufigsten ausgeführt?

[CPU]Central Processing Unit

[RAM] Random Access Memory

4.2 Standardmäßig verfügbare dezimale Genauigkeit

Der IEEE 754-1985 Standard für Gleitkommaarithmetik definiert zwei Formate zur Darstellung von Gleitkommazahlen. Diese besitzen jeweils eine fest vorgegebene Anzahl an verwendeten signifikanten Nachkommastellen. Diese sind Bestandteil der Programmiersprache C/C++ und werden als *float* und *double*-Datentypen bezeichnet. Die Tabelle 4.2 zeigt eine Übersicht der Basisdatentypen mit deren dezimalen Stellengenauigkeiten, Wertebereichen, Größen und Mantissenlänge[4]. Weiter ist dort auch der *long double*-Datentyp aufgelistet. Dieser wird nicht im IEEE-Standard definiert, wird aber aus Gründen der Vollständigkeit dennoch in die Vergleiche miteinbezogen.

Bezeichnung	**Größe**	**Dezimale Genauigkeit**	**Mantissenlänge**
float	4 Byte	6-stellig	23 Bit
double	8 Byte	15-stellig	53 Bit
long double	10 Byte	19-stellig	80 Bit

Tabelle 4.2: Übersicht der Wertebereiche und Stellengenauigkeiten der Standarddatentypen

4.3 Übersicht zu den verwendeten *multiprecision*-Bibliotheken

In Folgendem werden die verwendeten *multiprecision*-Bibliotheken kurz vorgestellt.

[4]vgl. dazu [Wolf06]

4.3.1 Die LiDIA - Bibliothek

Die LiDIA-Bibliothek[5] wurde an der Technische Universität Darmstadt[6] entwickelt. Ein Großteil der in der C-Standardbibliothek enthaltenen arithmetischen Grundoperationen ist in der LiDIA-Bibliothek auf Basis von Gleitpunktdatentypen implementiert. Zur Untersuchung wurde die aktuellste Version der LiDIA-Bibliothek verwendet (stand 18.05.2007, Version 2.2.0.). Die LiDIA-Bibliothek verwendet intern die GMP[7]. Außerdem konnten in Zusammenarbeit mit den Autoren der LiDIA-Bibliothek eine Reihe von Programmierfehlern beseitigt werden[8].

[GMP] GNU Multiprecision Library

4.3.2 Die NTL - Bibliothek

[NTL] Number Theorie Library

Die NTL – Bibliothek[9] wurde von Victor Shoup[10] an New York University[11] entwickelt. Sie ist eine hoch optimierte numerische Bibliothek, die einen ähnlich hohen Funktionsumfang aufweist, wie die LiDIA – Bibliothek. Sie bietet Datentypen für Fest – und Gleitkommaarithmetik mit einstellbarer Stellengenauigkeit. Darüber hinaus sind auch Vektor und Matrixoperationen enthalten. Jedoch ist die Auswahl an trigonometrischen Funktionen geringer (s. 4.4.2). Zur Untersuchung wurde die aktuellste Version der NTL-Bibliothek verwendet (Stand 18.05.2007, Version 5.4.1).

4.3.3 Die CLN - Bibliothek

[CLN] Class Library for Numbers

Die CLN - Bibliothek[12] wurde von Bruno Haible[13] entwickelt und wird derzeit von Richy Kreckel[14] betreut. Sie bietet eine Auswahl an elementaren mathematischen Funktionen für Gleitpunktarithmetik zur Rechnung mit beliebig einstellbarer Stellengenauigkeit. Nicht enthalten sind Vektor und Matrixoperationen. Zur Untersuchung wurde die aktuellste Version der CLN-Bibliothek verwendet (stand 18.05.2007, Version 5.4.1). Die CLN-Bibliothek verwendet intern die GMP . Es wurde die aktuellste Version der GMP verwendet (stand 18.05.2007, Version 4.2.1).

[GMP] GNU Multiprecision Library

4.3.4 Die MAPM - Bibliothek

Die MAPM - Bibliothek[15] wurde von Michael C. Ring[16] an der Universität von Minnesota[17] entwickelt. In der Bibliothek sind elementare mathematische Funktionen enthalten. Diese Funktionen arbeiten mit Fest – und Gleitkommazahlen, wobei die Stel-

[5] vgl. dazu [LiDIA]
[6] siehe [TUD]
[7] Es wurde die aktuellste Version der GMP verwendet (Veröffentlicht 2011-05-08, Version gmp-5.0.2).
[8] siehe mehr dazu unter http://www.cdc.informatik.tu-darmstadt.de/TI/LiDIA/#news, Download am 13.12.2011
[9] vgl. dazu [NTL]
[10] Victor Shoup: victor@shoup.net
[11] siehe [NYU]
[12] vgl. dazu [CLN]
[13] Bruno Haible: haible@clisp.cons.org
[14] Richy Kreckl: richard.kreckel@ginac.de
[15] vgl. dazu [MAPM]
[16] Michael C. Ring: mike.ring@honeywell.com
[17] siehe [UOM]

lengenauigkeit frei wählbar ist. Zur Untersuchung wurde die aktuellste Version der MAPM-Bibliothek verwendet (stand 18.05.2007, Version 4.9.2).

4.3.5 Die GMP - Bibliothek

[GMP] GNU Multi-precision Precision

Die GMP-Bibliothek ist eine weitere Bibliothek zur hochgenauen Rechnung. Sie bietet Unterstützung für zahlreiche Hardwareplattformen und Compiler. Wie viele andere Bibliotheken, wurde sie ursprünglich in der Programmiersprache C verfasst. Mittlerweile liegt eine C++-Adapter-Klasse vor, bei der alle gängigen Operatorüberladungen vorhanden sind. Für ausgewählte Hardwarearchitekturen wurde mit Hilfe von *inline-Assembler* die Geschwindigkeit erhöht. Für alle verwendeten Berechnungen wurde die Version *gmp-5.0.2* (veröffentlicht am 08.05.2011) verwendet.

4.4 Untersuchung des Funktionsumfangs

In den nachfolgenden Unterpunkten werden die in Abschnitt 4.3 vorgestellten Bibliotheken hinsichtlich ihres Funktionsumfangs gegenübergestellt. Dieser Vergleich bezieht sich lediglich auf *float*-Operationen. Um dem Vergleich übersichtlicher zu gestalten, wurden die Funktionen wie folgt gruppiert:

- **Grundrechenarten**
  Diese stellen Grundoperationen wie Addition, Subtraktion, Multiplikation und Division auf Basis des entsprechenden Datentyps dar.
- **Erweiterte mathematische Funktionen**
  In dieser Gruppe sind alle trigonometrischen, hyperbolischen, logarithmischen und exponentiellen Funktionen enthalten.

4.4.1 Basisoperationen

In Tabelle 4.3 werden die jeweils vorhanden Basisoperationen der hier untersuchten Bibliotheken dargestellt.

Funktion	Befehl	LiDIA	NTL	CLN	MAPM	GMP
Addition	+	✓	✓	✓	✓	✓
Subtraktion	-	✓	✓	✓	✓	✓
Multiplikation	*	✓	✓	✓	✓	✓
Division	/	✓	✓	✓	✓	✓
Quadratur	*quad*	✓	✓	✓	✓	-
Inversion	*invert*	✓	✓	✓	✓	-
$\sqrt[2]{x}$	*sqrt*	✓	✓	✓	✓	-
$\sqrt[3]{x}$	*cuberoot*	-	-	-	✓	-
Potenzieren	*pow*	✓	✓	-	✓	-
Betrag	*fabs*	✓	✓	✓	✓	✓
Signum	*signum*	-	-	✓	-	-

Tabelle 4.3: Vergleich der Basisoperationen

Alle betrachteten Bibliotheken besitzen Operatorüberladungen für Addition, Subtraktion, Multiplikation, Division und Betrag. Jedoch fehlen bei der GMP-Bibliothek alle weiteren grundlegenden Funktionen, wie etwa die Quadratwurzel. Die NTL - Bibliothek ist zusätzlich eine *cuberoot*-Funktion implementiert, welche in den anderen Bibliotheken fehlt. Die CLN - Bibliothek besitzt außerdem eine *Signum*-Funktion, welche in den anderen Bibliotheken nicht enthalten ist. Da bei der GMP-Bibliothek die grundlegenden mathematischen Transformationen nicht enthalten sind, wird sie für die folgenden Vergleiche ausgeklammert.

4.4.1.1 Kompatibilität mit den Namenskonventionen der mathematischen Standardfunktionen

Bei der Implementierung von Programmen, basierend auf *multiprecision*-Bibliotheken, fällt auf, dass oftmals die Namenskonventionen für die Standardfunktionen, wie sie in C/C++-Standardbibliothek[18] für mathematische Funktionen definiert sind, nicht oder nur bedingt eingehalten werden. Beispielsweise hat die Funktion zum Potenzieren (a^b) einer *bigfloat*-Zahl in der LiDIA-Bibliothek folgende Bezeichnung:

```
1   bigfloat power(const bigfloat &a, const bigfloat &b);
```

Listing 4.1: Power Funktion des bigfloat-Datentyps

Zum Vergleich ein Auszug aus der Standardbibliothek für den *double*-Datentyp:

```
1   double pow(const double &a, const double &b);
```

Listing 4.2: Power Funktion des double-Datentyps

Hier unterscheidet sich lediglich der Funktionsname. Würde der Funktionsname der LiDIA-Bibliothek *pow* anstatt *power*- heißen, dann wäre diese Funktion mit der mathematischen Standardbibliothek kompatibel. Diese kleinen Unterschiede erschweren die Implementierung von Algorithmen mit C++-Templates, da hierfür *wrapper*[19]-Funktionen geschaffen werden müssen.

4.4.2 Erweiterte mathematische Funktionen

In Tabelle 4.4 sind die jeweils verfügbaren transzendenten Funktionen eingetragen.

Funktion	**Befehl**	**LiDIA**	**NTL**	**CLN**	**MAPM**
Exponential	*exp*	✓	✓	✓	✓
Logarithmus	*log*	✓	✓	✓	✓
Sinus	*sin*	✓	✓	✓	✓
Kosinus	*cos*	✓	✓	✓	✓
Tangens	*tan*	✓	-	✓	✓
Kotangens	*cot*	✓	-	-	-
Arcus Sinus	*asin*	✓	-	✓	✓
Arcus Kosinus	*acos*	✓	-	✓	✓

[18] Siehe dazu in *math.h und cmath.h*

[19] Dies sind Funktionen, welche ein kompatibles Interface bereitstellen und ansonsten keinerlei Funktionalität besitzen.

Arcus Tangens	*atan*	✓	-	✓	✓
Arcus Kotangens	*acot*	✓	-	-	-
Sinus Hyperbolicus	*sinh*	✓	-	✓	✓
Kosinus Hyperbolicus	*cosh*	✓	-	✓	✓
Tangens Hyperbolikus	*tanh*	✓	-	✓	✓
Kotangens Hyperbolicus	*coth*	✓	-	-	-
Areasinus Hyperbolicus	*asinh*	✓	-	✓	✓
Areakosinus Hyperbolicus	*acosh*	✓	-	✓	✓
Areatangens Hyperbolicus	*atanh*	✓	-	✓	✓
Areacotangens Hyperbolicus	*acoth*	✓	-	-	-

Tabelle 4.4: Vergleich der transzendenten Funktionen

Die LiDIA-Bibliothek enthält alle Standardfunktionen und weist somit den größten Funktionsumfang auf. Ansonsten besitzen CLN und MAPM den selben Funktionsumfang. Am wenigsten Funktionen bietet die NTL - Bibliothek, in der keine Umkehr und Hyperbolikus - Funktionen implementiert sind. Diese könnten, falls nötig, mit Hilfe von Additionstheoremen nachgebildet werden. Davon soll hier jedoch abgesehen werden. Allein aus dieser Übersicht kann gefolgert werden, dass die LiDIA-Bibliothek als einzige geeignet ist, um mathematische Transformationen, wie sie in der Geodäsie und der Himmelsmechanik beispielsweise zu Bezugssystemtransformationen benötigt werden, mit hoher Anzahl signifikanter Nachkommastellen durchzuführen.

4.5 Untersuchung zur Abarbeitungsgeschwindigkeit

Im Folgenden sollen die *multiprecision*-Bibliotheken aufgrund ihrer Abarbeitungsgeschwindigkeit untersucht werden. Hierfür wurden synthetische Benchmarks entwickelt, um sowohl die Basisoperationen als auch die transzendenten Funktionen mit unterschiedlichen Genauigkeiten auf der TP1[20] miteinander vergleichen zu können.

[TP] Testplattform

Begriffsdefinition DEF4.1 „Synthetischer Softwarebenchmark“:
Ein synthetischer Softwarebenchmark ist ein Computerprogramm mit dem Zweck, die Leistungsfähigkeit unterschiedlicher Systeme bezüglich der Ausführungsgeschwindigkeit zu vergleichen. Bei einem Softwarebenchmark wird derselbe Algorithmus mit Hilfe verschiedener Bibliotheken ausgeführt und deren Ablaufzeiten gemessen. Die gemessenen Ablaufzeiten stellen vergleichbare Leistungsmerkmale der unterschiedlichen Bibliotheken dar.

4.5.1 Grundrechenarten

In diesem Abschnitt werden die vier Grundrechenarten einer Leistungsmessung unterzogen. Dabei wird bei einer jeweils vorgegebenen Genauigkeit eine bestimmte Grundrechenart (z.B. Addition) 10000 mal durchgeführt und der dafür benötigte zeitliche Aufwand gemessen. Außerdem wurde für jede der untersuchten Bibliotheken die Anzahl der signifikanten Nachkommastellen von 15 bis 150 sukzessive erhöht.

[20] Die genauen Leistungsmerkmale der Testplattformen sind in Abschnitt A.1 angegeben.

4.5.1.1 Addition

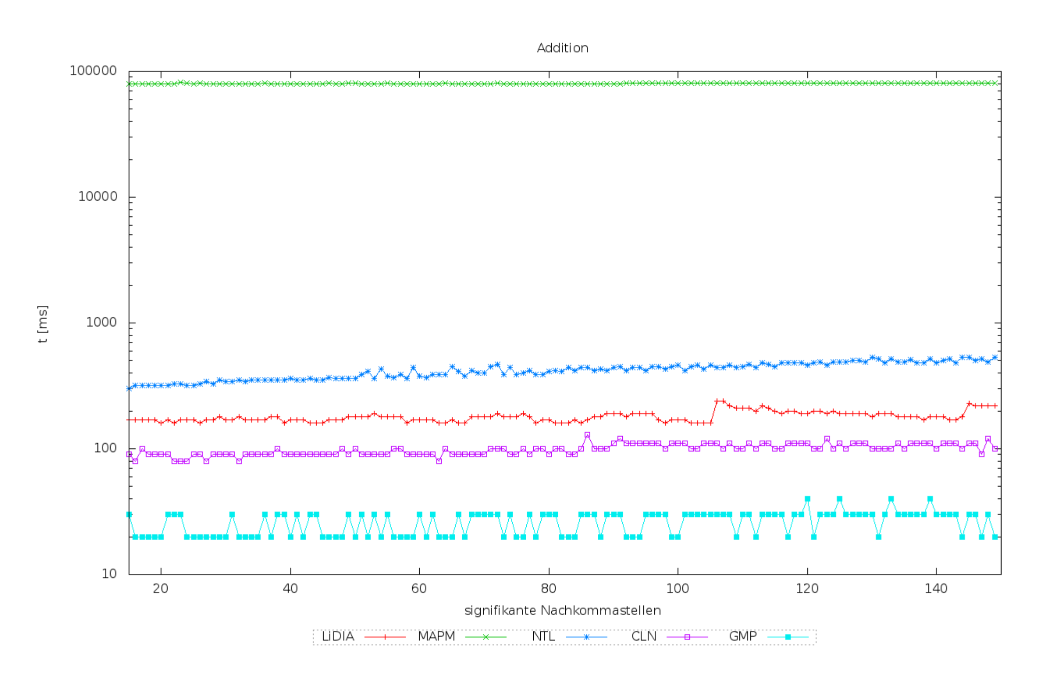


Abbildung 4.1: Leistungsmessungen der Addition auf Testplattform 1

Bei der Addition ist die MAPM-Bibliothek im gesamten Intervall um Faktor 10000 langsamer als die GMP-Bibliothek. Die NTL, LiDIA und CLN-Bibliotheken liegen bei allen untersuchten Genauigkeiten in einem Intervall zwischen 100 und 500 ms. Als schnellste Bibliothek geht bei der Additionsoperation die GMP-Bibliothek als bessere hervor.

4.5.1.2 Multiplikation

Beim Test zur Multiplikation zeigt sich ein ähnliches Bild wie bei der Addition. Die MAPM-Bibliothek ist deutlich langsamer als alle anderen Bibliotheken. Gegenüber der schnellsten Bibliothek (CLN) ist die MAPM-Bibliothek um den Faktor 10 langsamer. Die GMP und CLN-Bibliothek sind fast gleich schnell und die LiDIA-Bibliothek ist langsamer als die NTL-Bibliothek.

4.5.1.3 Subtraktion

Wird der Test zur Subtraktion betrachtet, so bietet sich ein ähnliches Bild wie bei den vorausgegangenen Untersuchungen. Die MAPM-Bibliothek ist um Faktor 10 langsamer als die schnellste (GMP) Bibliothek, NTL und CLN sind in etwa gleich schnell und die LiDIA-Bibliothek benötigt im Mittel 300 Millisekunden für einen Testdurchlauf.

4.5.1.4 Division

Beim Test zur Division ist, ganz im Gegensatz zu den vorausgegangenen Tests, die MAPM-Bibliothek am schnellsten. Die GMP-Bibliothek wird mit zunehmender An-

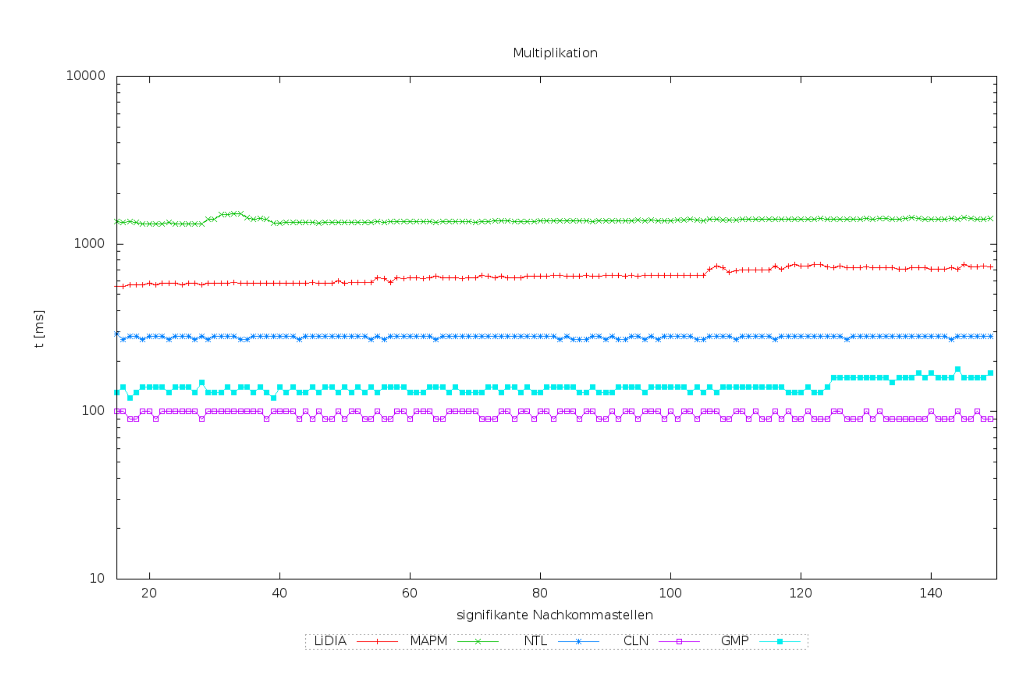


Abbildung 4.2: Leistungsmessungen der Multiplikation auf Testplattform 1

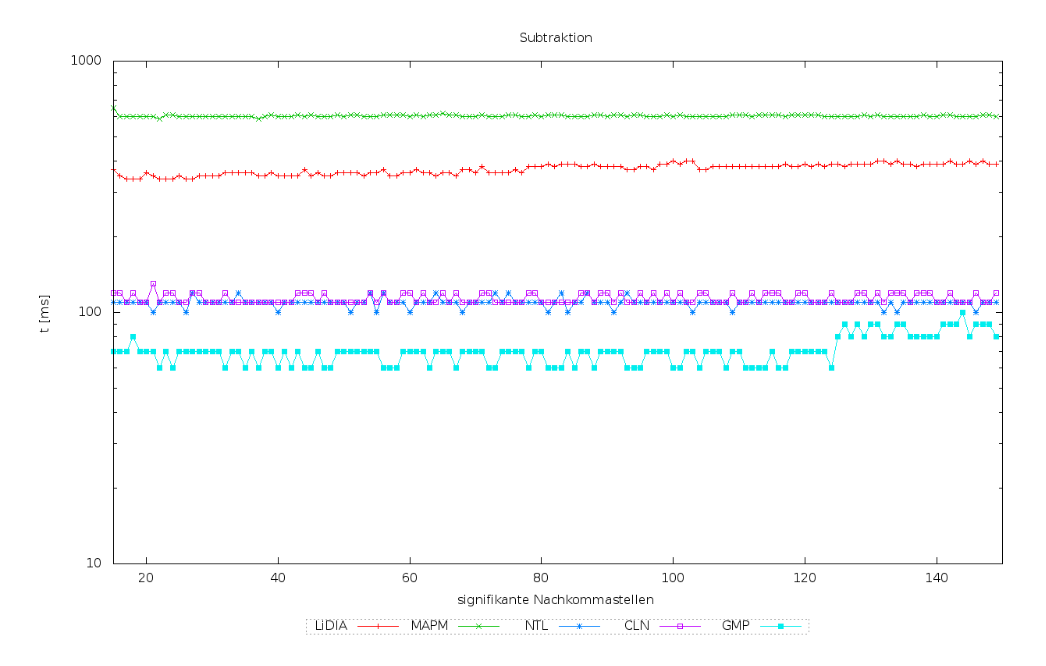


Abbildung 4.3: Leistungsmessungen der Subtraktion auf Testplattform 1

zahl signifikanter Nachkommastellen langsamer. Am langsamsten ist die LiDIA - Bibliothek. Die CLN- u. NTL - Bibliothek liegen nahe an der Geschwindigkeit der MAPM-Bibliothek und weisen durchwegs gut Geschwindigkeitswerte auf.

Allgemein betrachtet ist die MAPM-Bibliothek für Addition, Subtraktion und Multiplikation nicht geeignet, da alle anderen Bibliotheken bessere Leistungsdaten aufwei-

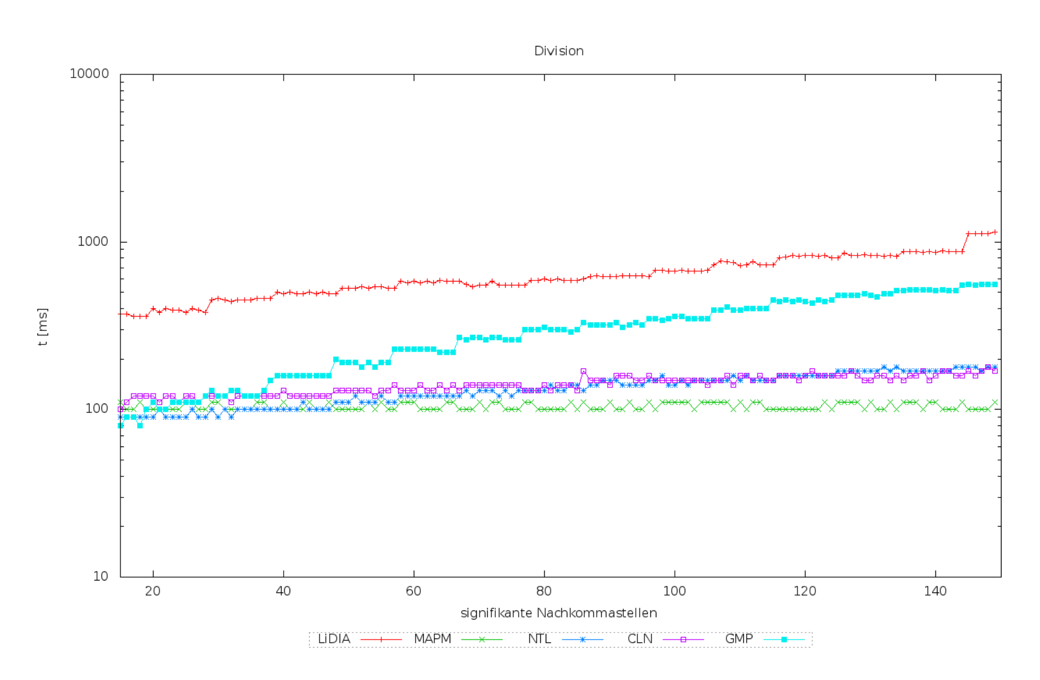


Abbildung 4.4: Leistungsmessungen der Division auf Testplattform 1

sen. Des Weiteren ist die GMP-Bibliothek, gefolgt von der CLN-Bibliothek, durchwegs sehr performant. Außerdem ist die LiDIA-Bibliothek in den meisten Fällen im hinteren Drittel angesiedelt. Weitere Ergebnisse zu Laufzeitmessungen der C++ *Inkrement* und *Dekrement* - Funktionen, in Abhängigkeit unterschiedlicher dezimaler Genauigkeiten, sind im Anhang F.5 zu finden.

4.5.2 Erweiterte mathematische Funktionen

Hier sollen die erweiterten mathematischen Funktionen einer Leistungsmessung unterzogen werden. Dazu wird analog zur vorausgegangen Untersuchung (Abschnitt 4.5.1) ein synthetischer Benchmark einer jeweiligen Funktion durchgeführt und deren Ausführungsgeschwindigkeit gemessen. Die Ergebnisse dieser Messungen für die Funktionen *log*, *sin*, *cos* und *tan* werden im Folgenden diskutiert. Die Resultate zusätzlicher Funktionen[21] sind in Abb. F.5 und F.6 (ab Seite 221) dargestellt. Bei dieser Untersuchung wurde die GMP-Bibliothek nicht einbezogen, da dieser die hierfür notwendigen Funktionen fehlen.

4.5.3 Logarithmus

Bei den Messungen der *log*-Funktion (siehe Abb. 4.5) ist die MAPM-Bibliothek über den gesamten Bereich um Faktor 100 langsamer als die CLN-Bibliothek. Die LiDIA-Bibliothek ist um Faktor 10 langsamer als die CLN-Bibliothek und signifikant langsamer als die NTL-Bibliothek. Insgesamt ist die CLN-Bibliothek bei diesem Versuch die schnellere. Auffällig sind die sprunghaften Anstiege bei der MAPM-Bibliothek im Bereich von 30 und 120 signifikanten Nachkommastellen und bei der LiDIA-Bibliothek

[21] *exp,sqrt,asin,acos,atan,sinh,cosh,tanh*

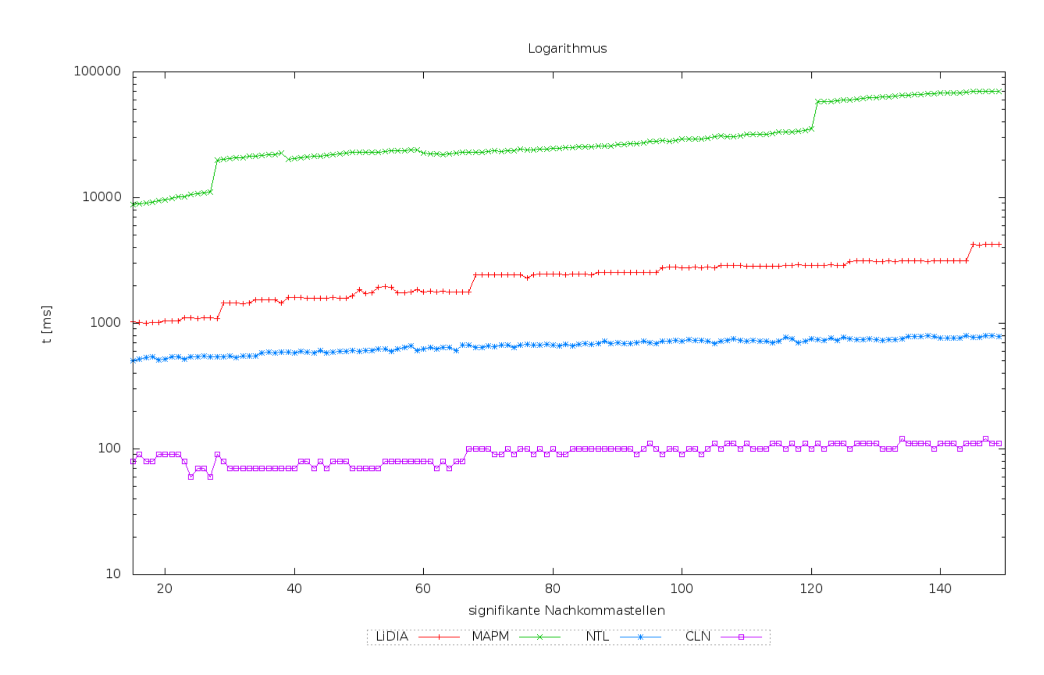


Abbildung 4.5: Ergebnisse der Laufzeitmessungen der *log*-Funktion (Testplattform 1)

im Bereich von 16, 70 und 150 Nachkommastellen.

4.5.3.1 Sinus

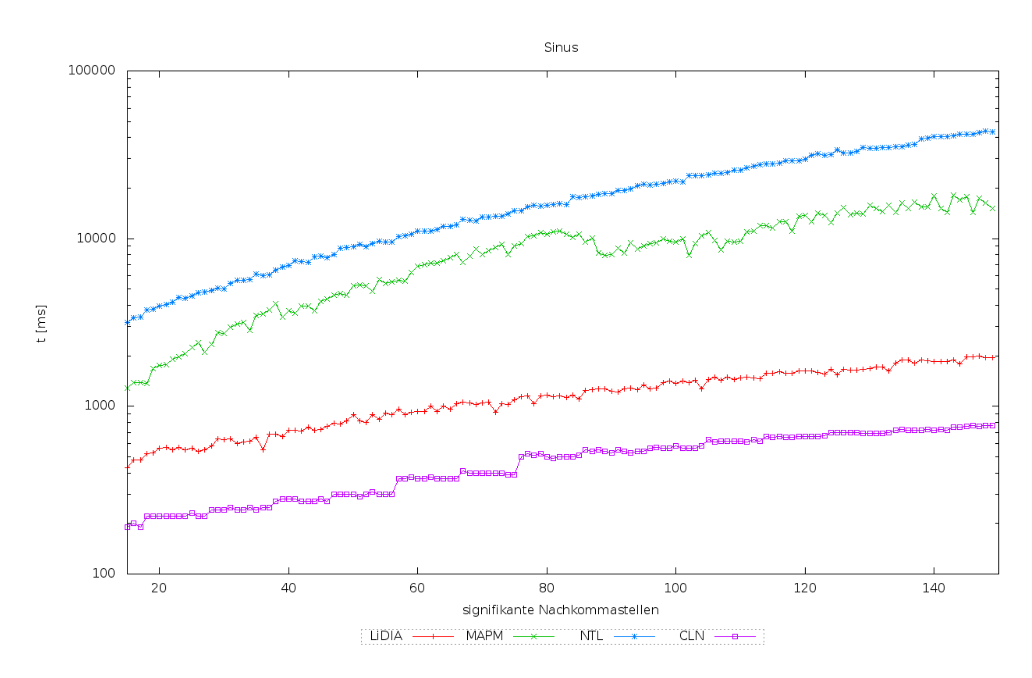


Abbildung 4.6: Ergebnisse der Laufzeitmessungen der *sin*-Funktion (Testplattform 1)

Werden die Laufzeitmessungen (siehe Abb. 4.6) genauer betrachtet, so fällt auf, dass bei allen Bibliotheken die Laufzeit mit zunehmender Anzahl signifikanter Nach-

kommastellen steigt. Dennoch erscheint die CLN - Bibliothek am schnellsten, gefolgt von der LiDIA-Bibliothek. Besonders langsam ist in diesem Fall die NTL-Bibliothek. Diese ist um Faktor 100 langsamer als die CLN-Bibliothek.

4.5.3.2 Kosinus

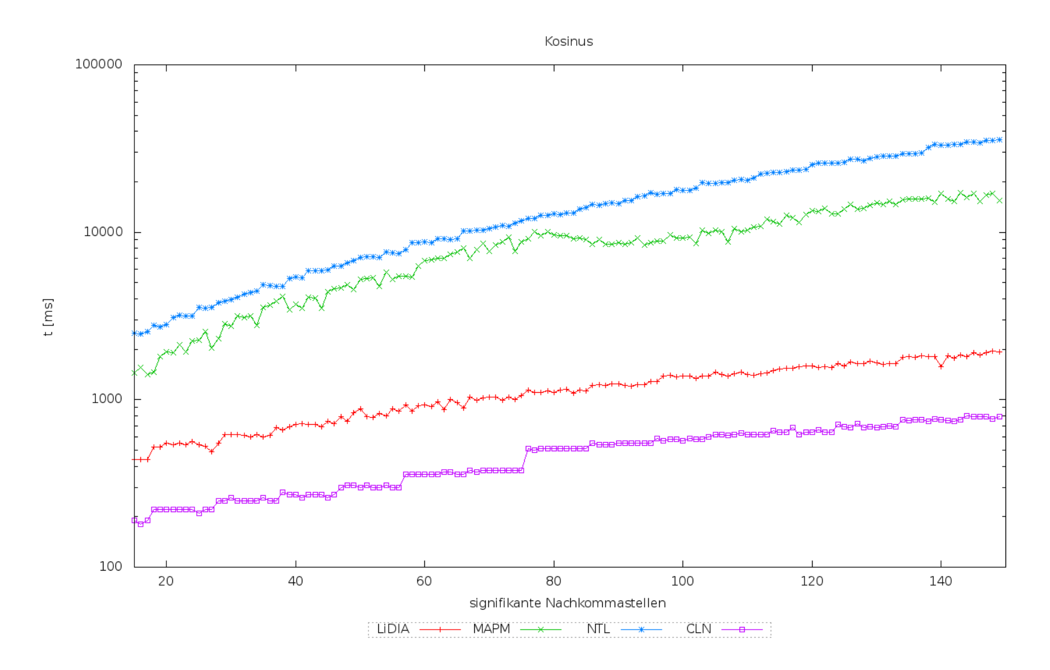


Abbildung 4.7: Ergebnisse der Laufzeitmessungen der *cos*-Funktion (Testplattform 1)

Die Messungen zur Kosinusfunktion (siehe Abb. 4.7) ist, verglichen mit den Messungen zur Sinusfunktion nahezu identisch. Hierbei zeigt sich wieder die Schwäche der NTL und MAPM-Bibliothek hinsichtlich trigonometrischer Funktionen.

4.5.3.3 Tangens

Die Laufzeitmessung zur Tangensfunktion (siehe Abb. 4.8) wurde ohne die NTL-Bibliothek durchgeführt, da diese keine *tan*-Funktion bereitstellt. Hierbei ist die CLN-Bibliothek im gesamten Bereich um Faktor 100 schneller als die MAPM-Bibliothek.

4.6 Fazit

Die Frage, welche Bibliothek am geeignetsten für diese Arbeit ist, lässt sich nicht pauschal beantworten. Würde ausschließlich die Abarbeitungsgeschwindigkeit eine Rolle bei der Auswahl spielen, dann wäre die CLN-Bibliothek die erste Wahl. Diese besitzt jedoch, verglichen mit der C/C++-Standardbibliothek, nicht alle notwendigen mathematischen Transformationen, die für diese Arbeit notwendig sind. Außerdem kann bei der CLN-Bibliothek die jeweilig verwendete numerische Genauigkeit nicht direkt eingestellt werden, d.h. die numerische Genauigkeit wird automatisch bei der Initialisierung von Variablen gesetzt. Dies wird an dieser Stelle als weiterer Nachteil gewertet.

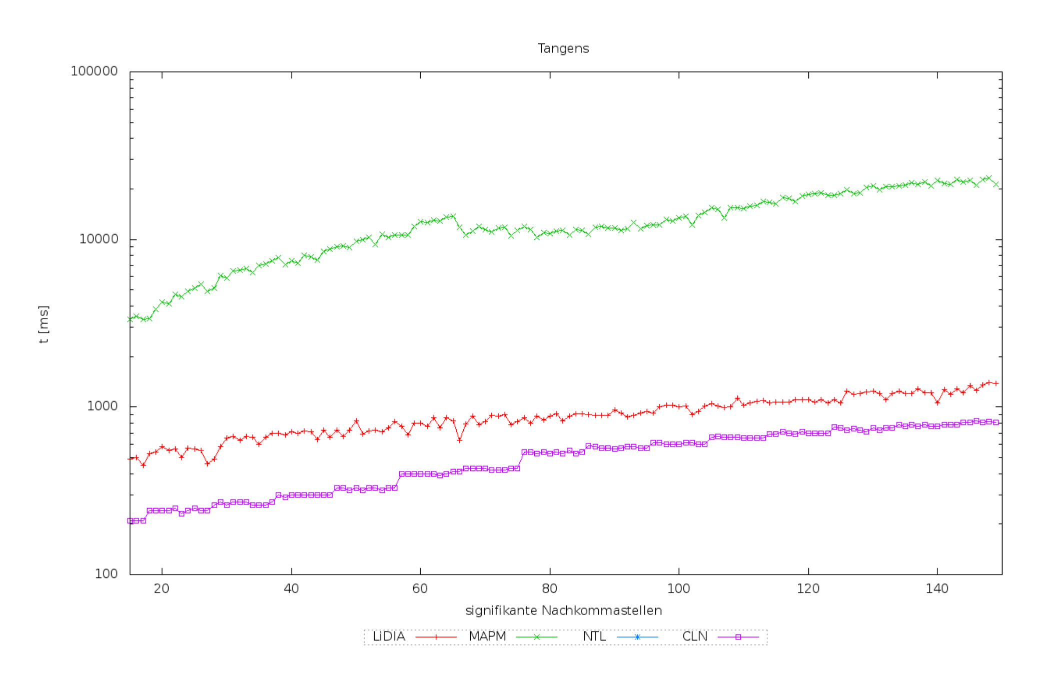


Abbildung 4.8: Ergebnisse der Laufzeitmessungen der *tan*-Funktion (Testplattform 1)

Die LiDIA-Bibliothek ist beim Laufzeitvergleich in vielen Fällen hinter der CLN-Bibliothek, besitzt jedoch den vollen Funktionsumfang der C/C++-Standardbibliothek. Besonders bei den trigonometrischen Umkehrfunktionen (siehe F.11), welche häufig bei Bezugssystemtransformationen benötigt werden, ist die LiDIA-Bibliothek schneller. Abgesehen von kleineren Abweichungen in der Nomenklatur der Standardfunktionen erscheint diese Bibliothek am geeignetsten.
Die hierfür angestellten Vergleiche decken nur einen Bruchteil der verfügbaren Funktionen der untersuchten Bibliotheken ab. Außerdem wurden nur zwei Kriterien zur Validierung herangezogen. Weitere mögliche Kriterien wären:

- Welchen Umfang und Qualität besitzt die Dokumentation der jeweiligen Bibliothek.
- Wurde die untersuchte Bibliothek weiterentwickelt und in anderen Projekten aktiv eingesetzt?
- Ist die Bibliothek für eine bestimmte Plattformen optimiert?
- Wie aktuell ist die bestehende Version, d.h. wann war die letzte Aktualisierung?
- Gibt es ein Forum, in dem die Anwender Fragen an die Entwickler stellen können?
- Ist die Bibliothek mit den aktuellen Compilern kompatibel?
- Für welche Plattformen ist die Bibliothek verfügbar?
- Unterstützt die Bibliothek bereits Optimierungen für 64-Bit Arithmetik?

Die Beantwortung dieser Fragen würde den Rahmen dieser Arbeit sprengen.
Außerdem könnten zur Leistungsmessung, ganz im Gegensatz zur Laufzeitmessung, auch die Anzahl der Prozessortaktzyklen aufgezeichnet werden und diese als weiteres Qualitätskriterium herangezogen werden. Hierzu wird auf [Agner] verwiesen, dort stehen die nötigen Softwareroutinen und umfangreiches Informationsmaterial zur Messung der Taktzyklen und zur Programmoptimierung bereit.
Weiterhin wird noch auf die Veröffentlichung von [Zimmermann98] hingewiesen, worin *multiprecision*-Bibliotheken und CAS u.a. aufgrund ihrer Abarbeitungsgeschwindigkeit bei der Integerrechnung miteinander verglichen werden.

[CAS] Computer Algebra Systeme

Kapitel 5

Methoden zur Laufzeitverbesserung von Programmen

Schwerpunkt des Kapitels:

Ziel dieses Abschnitts ist es, Strategien zur Programmoptimierung für Desktop-Computer und Workstations bereitzustellen. Diese sind, verglichen mit Hochleistungsrechnern, nicht so leistungsfähig, aber weiter verbreitet und stehen folglich einer größeren Nutzergemeinde zur Verfügung.
Hierfür wird der Entwicklungsprozess eines Programms kategorisiert und in mehrere Phasen unterteilt. In jeder dieser Entwicklungsphasen gibt es gewisse Techniken, um die Ablaufgeschwindigkeit eines Programms auf einer bestimmten Zielarchitektur zu beeinflussen. Diese werden erst beschrieben und anschließend anhand einschlägiger Fallbeispiele erklärt. Grundlegend soll zwischen folgenden Methoden zur Laufzeitverbesserung unterschieden werden:

- C++ Templatemetaprogrammierung [TMP]
- Einsatz von Hilfsmitteln der Parallelverarbeitung mit OpenMP
- Optimierung durch den Compiler

5.1 Verschiedene Arten der Optimierung

Soll ein Programm hinsichtlich Abarbeitungsgeschwindigkeit optimiert werden, so kann dies in verschiedenen Phasen der Programmentwicklung durchgeführt werden. Der gesamte Entwicklungsprozess, bis hin zur Laufzeit, kann in drei verschiedene Phasen unterteilt werden. Am Anfang steht die Entwicklungsphase, in welcher das Programm in einer bestimmten Programmiersprache erstellt wird. Daraufhin folgt die Übersetzungsphase, in der das Programm in die Maschinensprache übersetzt wird, gefolgt von der Laufzeitphase, in der das Programm ausgeführt wird. Diese Phasen sind in Abb. 5.1 dargestellt.

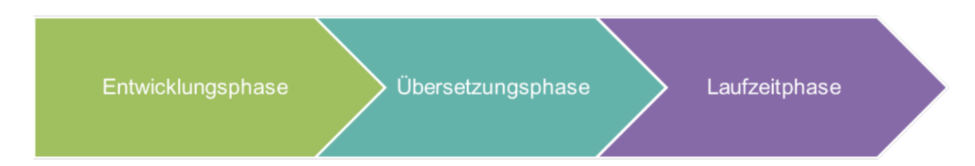


Abbildung 5.1: Entwicklungsphasen bei der Erstellung eines Programms

Während der Entwicklungsphase gibt es verschiedene Techniken und Vorgehensweisen, die helfen, ein Programm zu optimieren. Diese können weiter kategorisiert werden und stellen hohe Anforderungen an den Programmierer. Dieser muss an der entsprechenden Stelle auf die richtige Methode zurückgreifen, um ein Optimum zu erreichen. Hierbei wird darauf geachtet, dass beispielsweise unnötige Rechenoperationen eingespart werden und somit die Abarbeitungsgeschwindigkeit des optimierten Programms erhöht wird. Viele dieser Techniken, speziell für die Programmiersprachen C und C++, sind in [Agner] zu finden. Im Falle der Programmiersprachen C und C++ ist es gängige Praxis, dass Teile des Quellcodes in der Maschinensprache Assembler[1] eingefügt werden, um somit das Programm an eine bestimmte Hardware anzupassen. Des Weiteren gibt es spezielle Techniken der Templatemetaprogrammierung[2], mit der die Geschwindigkeit eines Programms ebenfalls erhöht werden kann. Außerdem können viele Problemstellungen parallelisiert werden und dadurch weitaus höhere Geschwindigkeiten erreichen, als ein seriell abarbeitendes Programm. Dazu gibt es für die Programmiersprachen C und C++ die zwei verschiedenen Techniken OpenMP[3] und MPI[4].
Ferner wird das erstellte Programm in der Übersetzungsphase einem Compiler übergeben, der weitere Optimierungen durchführt und das Programm an die jeweilige Hardwareplattform anpasst. Dabei gibt es eine Vielzahl von Optimierungsoptionen, die während des Übersetzens aktiviert werden können. Diese bringen auf manchen Plattformen bei gewissen Programmen mehr Geschwindigkeit, jedoch kann keine umfassende Garantie dafür gegeben werden. Tatsache ist, dass sich manche Optimierungsoptionen sogar nachteilig auf die Laufzeit auswirken. An dieser Stelle bleibt dem Programmierer oft nichts anderes übrig, als die Dokumentation des Compilers zu studieren. Um diesen Prozess etwas zu vereinfachen wird auf den nächsten Unterpunkt

[1] siehe mehr dazu unter [Backer03]
[2] siehe mehr dazu unter [Stroustrup04]
[3] siehe mehr dazu unter [Hoffmann08],[Rauber08] und [OpenMP]
[4] siehe mehr dazu unter [Rauber08] und [MPIStandard]

hingewiesen, in welchem eine Möglichkeit dargestellt wird, die benötigten Optimierungsoptionen automatisiert zu bestimmen.
Nachdem das Programm übersetzt und optimiert ist, kann es ausgeführt werden (Laufzeitphase). Besonders bei numerischen Berechnungen ist es oft wichtig, die Zeitspanne zwischen Start und Beendigung des Programms so gering wie möglich zu halten. Soll nun ein Programm optimiert werden, so muss festgestellt werden, welcher Programmteil die meiste Rechenzeit in Anspruch nimmt. Dies kann mit sogenannten *profiling*-Programmen durchgeführt werden. Diese messen beispielsweise während der Laufzeitphase die Anzahl und Zeitdauer bestimmter Funktionsaufrufe und erstellen daraus ein Protokoll. Aus diesem kann der Programmierer die Stellen im Programm identifizieren, welche die meiste Laufzeit beanspruchen und diese verbessern. Im Rahmen dieser Arbeit wurde meist das *profiling*-Programm *gprof*[5] aus der GNU-Compiler-Collection verwendet. Die Vorgehensweise und der Umgang mit diesem hilfreichen Werkzeug soll anhand eines Beispiels gezeigt werden:

Beispiel 5.1 „Identifikation von zeitintensiven Programmstellen“
Um zeitintensive Programmstellen zu identifizieren, muss das Programm zunächst übersetzt werden. Dabei ist es erforderlich, eine bestimmte Compileroption ($-pg$) zu aktivieren. Im Anschluss kann das Programm gestartet werden. Dabei wird die jeweilige Anzahl Funktionsaufrufe und deren Zeitdauer mitprotokolliert und in einer Datei abgelegt. Diese Datei heißt standardmäßig $gmon.out$ und kann dann dem Hilfsprogramm gprof übergeben werden, das aus dem Protokoll und dem erstellten Programm einen Analysereport generiert. Für ein Programm zur Integration des Duffing Oszillators mit Hilfe des Potenzreihenintegrators unter Verwendung des Decimal-Datentyps konnte folgender Analysereport (siehe Tabelle 5.1) erstellt werden. Hierbei wird lediglich ein kurzes Extrakt des Analysereports in kompakter Form gezeigt.

Zeit [%]	***Zeit [s]***	***Aufrufe***	***Funktionsname***
37.93	*0.11*	*186219*	$Decimal :: mul(...)$
20.69	*0.06*	*672403*	$Decimal :: _rshift(...)$
6.90	*0.02*	*670397*	$Decimal :: _round(...)$

Tabelle 5.1: Wichtige Ergebnisse der gprof Analyse

Hierbei ist in der ersten Spalte der prozentuale Anteil der jeweiligen Funktion zur Laufzeit angegeben. Die Funktion $Decimal :: mul(...)$ nimmt 37,93% der Gesamtlaufzeit in Anspruch, wobei alle Funktionsaufrufe zusammen 0.11 Sekunden Laufzeit beansprucht haben (Spalte 2). Außerdem wurde diese Funktion 186219 mal aufgerufen (Spalte 3). Aus diesen wenigen Zeilen lassen sich einige Erkenntnisse gewinnen:

- *Mögliche Optimierungen in den Funktion $Decimal :: mul(...)$ und $Decimal :: _rshift(...)$ sind lohnenswert, da diese relativ viel Laufzeit beanspruchen (zusammen über 50%).*
- *Ferner wird die Funktion $Decimal :: _rshift(...)$ sehr oft aufgerufen, benötigt aber weniger Laufzeit als die Funktion $Decimal :: mul(...)$. Dadurch könnte bei Reduktion der Aufrufe von $Decimal :: _rshift(...)$ unnötige Laufzeit eingespart werden.*

Wie in diesem Beispiel gezeigt, stellt ein *profiling*-Programm ein wertvolles Werkzeug zur Optimierung dar. Weitere *profiling*-Programme sind u.a. [CodeAnalyst] und [IntelVTune].

[5] siehe mehr dazu unter [GProf]

5.2 Hard - und Softwareplattform

Begriffsdefinition DEF5.1 „Plattform“:
Der Begriff Plattform ist ein Überbegriff der Informatik und steht für eine bestimmte Zusammenstellung aus Rechnerarchitektur, Programmiersprache und den dabei verwendeten Bibliotheken mit deren Laufzeitumgebung. Damit kann verschiedenen Konstellationen, bestehend aus Hard. u- Softwarekomponenten, ein Name zugeordnet werden.

Dieser abstrakte Begriff kann weiter verfeinert werden, indem zwischen den verwendeten Hard- u. Softwarekomponenten unterschieden wird.

Begriffsdefinition DEF5.2 „Hardwareplattform“:
Der Begriff Hardwareplattform definiert die verwendete Prozessorarchitektur und die verwendete Hardware im Allgemeinen. Hiermit wird beispielsweise eine einheitliche Maschinensprache, Byte-Reihenfolge und Datenwortgröße festgelegt.

Begriffsdefinition DEF5.3 „Softwareplattform“:
Durch Festlegung auf bestimmte Softwarekomponenten, bestehend aus Betriebssystem, Programmiersprache und Laufzeitumgebung, wird eine bestimmte Softwareplattform definiert.

Aufgrund der Vielzahl verfügbarer Hard- und Softwareplattformen muss eine Einschränkung hinsichtlich der hier aufgezeigten Methoden zur Laufzeitverbesserung getroffen werden: Alle durchgeführten Methoden führen in vielen Fällen zu Laufzeitverbesserungen auf Systemen mit x86-Prozessorarchitektur. Sie wurden auf Single- und Dual-Core-Prozessoren mit 32/64 Bit-Datenwortgröße getestet. Ferner wurden die entwickelten Programme auf Linux basierenden Betriebssystemen[6] (Ubuntu) in der Programmiersprache C und C++ entwickelt. Damit die Laufzeitmessungen wiederholbar sind, befinden sich die Eckdaten der verwendeten Testplattformen in Abschnitt A.1 (Seite 183). Folglich kann keine Aussage über das Laufzeitverhalten dieser Optimierungsmethoden auf anderen Plattformen getroffen werden.

5.3 Optimierung während der Entwicklungsphase

Es gibt viele Möglichkeiten, die resultierende Laufzeit eines Programms während der Entwicklungsphase zu beeinflussen. Dabei können die jeweiligen Maßnahmen in mehrere Gruppen eingeteilt werden:

- **Allgemeine Optimierungstechniken**
  Hier können eine Reihe von Faustregeln angegeben werden, welche helfen, die Laufzeit eines Programms zu beschleunigen.

  1. Jeder Zugriff auf das Dateisystem benötigt kostbare Laufzeit, damit sollte demnach sparsam umgegangen werden.
  2. Jeder Funktionsaufruf kostet Laufzeit. Aus diesem Grund sollten kleine Funktionen, die häufig aufgerufen werden, mit einem *inline* - Schlüsselwort versehen werden.

[6] Folgende Compiler wurden verwendet: GCC, clang und Intel-Compiler.

3. Das Kopieren von Objekten kostet unnötige Laufzeit. Aus diesem Grund sollten Referenzen bzw. konstante Referenzen bevorzugt verwendet werden. Ist es dennoch nötig ein Objekt zu kopieren, dann sollte der Konstruktor und nicht der Zuweisungsoperator verwendet werden, da letzterer i.d.R. mehr Funktionsaufrufe und folglich mehr Laufzeit beansprucht.
4. Jede Berechnung bzw. Vereinfachung, die schon in der Entwicklungsphase durchgeführt werden kann, sollte schon während dieser durchgeführt werden. Wird die Berechnung in die Übersetzungsphase verlagert, d.h. dem Compiler überlassen, steigt die Übersetzungszeit u.U. unnötig an.
5. Das Anlegen einer Variablen kostet Laufzeit, da hierfür Speicher angefordert werden muss. Aus diesem Grund ist es empfehlenswert, unnötige Variablen zu sparen.

Diese Techniken können natürlich weiter ausgebaut werden. In [Bulka99] wird anhand vieler Beispiele aufgezeigt, wie aufgrund der bereits gezeigten Techniken (und mehr) Geschwindigkeitsengpässe vermieden werden können. Außerdem wird vor möglicher falscher Herangehensweise zur Optimierung gewarnt. Hier ein kurzer Auszug aus [Bulka99] (Seite 134):

80-20 Rule: Speed Up the Common Path

The 80-20 rule has many applications: 80% of the execution scenarios will traverse only 20% of your source code, and 80% of the elapsed time will be spent in 20% of the functions encountered on the execution path. The 80-20 rule is the dominating force driving the argument that premature tuning is a sin. If you randomly tune everything you can think of, not only do you waste 80% of the effort, you will also hack the design beyond repair.

- **Spezielle Optimierungstechniken**

  - **Parallelverarbeitung**
    Falls ein Problem in verschiedene parallele Aufgaben unterteilt werden kann, bieten sich die beiden Parallelisierungstechniken OpenMP[7] und MPI[8] an. Mit ersterer können sowohl Schleifen als auch Funktionsblöcke parallelisiert werden. Diese Art der Parallelisierung wird auch als *shared memory*-Parallelisierung bezeichnet, da hierbei nur ein Programmspeicher für alle parallel arbeitenden Ablauffäden (*threads*) vorhanden ist. Ganz im Gegensatz zum MPI, denn dort wird die Parallelität mit Hilfe von Prozessen realisiert. Diese besitzen ihren eigenen Speicherbereich und können auch auf verschiedenen Computern laufen. Darum wird MPI auch als *distributed memory*-Parallelisierung bezeichnet.

    [MPI] Message Passing Interface

  - **Metaprogrammierungstechniken**
    Der Begriff Metaprogrammierung steht i.A. für Programmiertechniken,

[7] siehe mehr dazu unter [OpenMP]
[8] siehe mehr dazu unter [MPIStandard]

bei denen aus einem Quelltext wieder Quelltext erzeugt bzw. generiert wird. Unter Templatemetaprogrammierung [TMP] versteht man einen speziellen Teil der Programmiersprache C++. Hierbei kann mit Hilfe der Sprache C++ wiederum C++-Quelltext generiert werden. Dies ist nicht mit Makroprogrammierung aus der Programmiersprache C zu verwechseln, da hier überwiegend C++-Templates verwendet werden und diese während der Übersetzungsphase einer syntaktischen Prüfung unterzogen werden. Damit ist es möglich, den Präprozessor eines C++-Compilers zur Quelltextgenerierung zu verwenden, um in der Übersetzungsphase einen Quelltext zu generieren, der das zugrunde liegende Problem effizienter löst. Diese Art der Programmierung bietet also die Möglichkeit, ein Programm zu schreiben, welches sich selbst verändert, bevor es zur Ausführung (Laufzeitphase) kommt.

– **Einsatz von Maschinensprache (Assembler)**
  Diese Optimierungstechnik kann verwendet werden, um an gewissen Stellen im Quellcode direkt in Maschinensprache zu programmieren. Sie wird i.A. nur angewendet, falls die Optimierung des Compilers nicht das gewünschte Resultat erbringt. Allerdings birgt diese Art der Optimierung auch einige Tücken, da durch die Verwendung der Maschinensprache die Software auf eine bestimmte Hardwareplattform optimiert wird, was zu Portabilitätsproblemen führen kann. Somit kann nicht garantiert werden, dass dieses Programm auf einer anderen Hardwareplattform ein ähnliches Laufzeitverhalten besitzt. Soll beispielsweise eine Subroutine in Maschinensprache programmiert werden, so gilt es, einen optimalen Satz an Maschinensprachbefehlen zu finden, der das Problem auf der jeweiligen Hardwareplattform möglichst effizient löst. Hierbei können sogenannte *superoptimizer*[9] helfen. Diese sind Programme, die durch exzessives Suchen und Messen des Laufzeitverhaltens den optimalen Satz an Maschinensprachbefehlen ermitteln.

5.3.1 C++-Templatemetaprogrammierung [TMP]

Bei dieser Programmiertechnik wird darauf geachtet, möglichst alle Informationen, welche für die Übersetzungsphase festgelegt und somit statisch im Quelltext (Metadaten) vorhanden sind, auszunutzen um daraus das Programm zu verändern. Hierbei soll die Veränderung in einem schnelleren und besser optimierten Programm liegen. Diese Technik der gezielten Ausnutzung statischer Informationen wurde 1994 von Erwin Unruh entdeckt[10].

Begriffsbestimmung BEG5.1 „Metadaten“:
Dies sind Daten, die während der Übersetzungsphase manipuliert werden können, d.h. Typen, Referenzen, Zeiger, konstante Elemente und Klassenelemente.

5.3.1.1 Arbeitsweise eines C++-Templatemetaprogramms

[9] siehe mehr dazu unter [Joshi01], [Brain06] und [Bansal06]
[10] siehe mehr dazu unter [Unruh94]

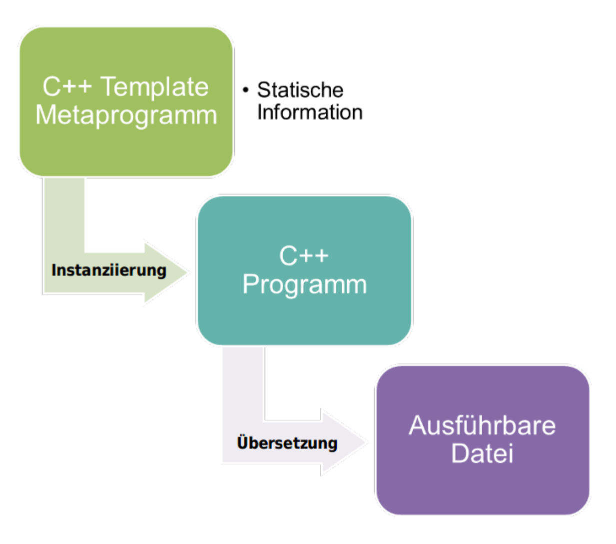


Abbildung 5.2: Schematischer Ablauf zum Übersetzungsvorgang eines C++-Templatemetaprogramms

Der Übersetzungsvorgang eines C++-TMP kann in zwei aufeinander folgende Phasen unterteilt werden, was in Abb. 5.2 illustriert ist:

[TMP] Templatemetaprogramm

1. Im ersten Schritt erfolgt die Instanziierunsphase. Dabei werden die erzeugten Templates durch den C++-Präprozessor interpretiert. In dieser Phase erfolgt die Veränderung des Programms, da der Präprozessor bestimmte Konstrukte rekursiv ersetzt. Durch Ausnutzung dieser Eigenschaft kann sich ein Programm während der Übersetzungsphase selbst umschreiben.

2. Anschließend erfolgt die Interpretation mit darauf folgender Umwandlung in Maschinencode. Dies geschieht analog zum Übersetzen eines herkömmlichen C++ Programms.

```
1  template <typename T> void Square(VectorT<T> &a)
2  {
3    VectorT<T> v(a);
4    for(unsigned int n=0; n < v.uiSize(); n++)
5    {
6      for(unsigned int i=0; i <= n; i++)
7      {
8        a[n] += v[i] * v[n-i];
9      }
10   }
11 }
```

Listing 5.1: Square Funktion (Version 1)

Beispiel 5.2 für „Anwendung von TMP auf Schleifen“
In diesem einführenden Beispiel zur C++-Metaprogramierungs-Technik soll kurz die Grundidee anhand der Vereinfachung von Schleifen während der Übersetzungsphase gezeigt werden. Diese Technik wird auch bei Compilern zur Optimierung eingesetzt und dort als loop-unrolling *bezeichnet. In Listing 5.1 wird ein Ausschnitt einer C++ Funktion gegeben, welche das Quadrat einer gegebenen Potenzreihe berechnet. Der formelmäßige Zusammenhang zur Quadratur*

einer Potenzreihe ist in Abschnitt 2.4.2 ab Seite 31 angegeben. Die hier verwendete Funktion wurde für dieses Beispiel vereinfacht und entstammt dem PowerSeriesT-Modul. Werden die beiden for-Schleifen in den Zeilen 4 und 6 betrachtet, so iterieren diese jeweils über eine bestimmte Anzahl von VectorT<T>[11]-Elementen. Die Anzahl der VectorT<T>-Elemente ist in diesem Fall nicht statisch vorhanden und wird demnach zur Laufzeitphase abgefragt. Um diese Funktion in ein Templatemetaprogramm zu wandeln ist es im ersten Schritt erforderlich, die Anzahl der VectorT<T>-Elemente zur Übersetzungszeit zu kennen. In einer leicht veränderten Version (siehe Listing 5.2) wurde die VectorTMP-Klasse[12] verwendet, welche die Anzahl der Elemente (N) zur Übersetzungsphase statisch zur Verfügung stellt[13]. Durch diese Modifikation werden dem Compiler die nötigen Informationen bereit gestellt, um einen besser optimierten Maschinencode zu erzeugen. Somit ist es nicht mehr nötig, die beiden for-Schleifen während der Laufzeitphase iterieren zu lassen. In Version 2 wird diese Aufgabe dem Compiler überlassen, welcher je nach Optimierungseinstellungen mehr oder weniger stark optimiert. Soll sichergestellt werden, dass die Schleifen während der Übersetzungsphase vereinfacht werden, so kann man dies mit rekursiv definierten Template-Metafunktionen gewährleisteten. Hierzu wurde eine weitere Version (siehe Listing 5.3) erstellt, welche auf der zweiten Version basiert, jedoch mit Hilfe von Template-Metprogrammierung erzwingt, dass die äußere Schleife während der Übersetzungsphase vereinfacht wird. Da sich die Template-Metaprogrammierungstechnik von der herkömmlichen C++-Templateprogrammierung in vielen Punkten unterscheidet, soll hier die Funktionsweise der Version 3 erklärt werden:

- Zeile 13-18
  In den Zeilen 13 und 14 ist analog zu den Versionen 2 und 3 die Funktion Square definiert. In Zeile 15 wird eine Kopie des VectorTMP<T>-Objects angelegt und in Zeile 17 erfolgt der Aufruf der äußeren for-Schleife, welche hier als OuterLoop bezeichnet wird. Diese Funktion enthält zwei Template-Parameter: an erster Stellen den Datentyp (T) und zweitens den Startindex der Schleife mit dem Wert 0. Außerdem besitzt diese Funktion zwei Funktionsparameter vom Typ VectorTMP<T> (siehe mehr dazu in Zeile 1-3).

- Zeile 1-3
  Dies ist die Funktionsdeklaration der Funktion OuterLoop mit den beiden Template-Parametern T und n und den Funktionsparametern vom Typ VectorTMP<T>. Hierbei ist der zweite Parameter als konstante Referenz definiert, d.h. der Wert dieses Objekts kann innerhalb dieser Funktion nicht verändert werden, was eine Art Schreibschutz darstellt und dem Compiler zusätzlich bei der Optimierung hilft.

- Zeile 5-7
  In den Zeilen 5 und 6 wird die innere Schleife abgearbeitet. Hierbei gibt es keinen Unterschied zu den anderen beiden Versionen. In Zeile 7 wird die Funktion OuterLoop erneut aufgerufen. Diese erzwingt eine Rekursion während der Instanziierunsphase, welche Teil der Übersetzungsphase ist. Dabei erfolgt der Aufruf mit einem modifizierten Index, analog zum Ablauf zur Arbeitsweise einer for-Schleife.

- Zeile 9-11
  Hier wird die Funktion erneut definiert, jedoch in spezieller Form. Damit die Rekursion aus Zeile 5 bis 7 terminieren kann, wird ein Abbruchkriterium benötigt. Dies geschieht, indem die Funktion OuterLoop partiell spezialisiert wird und für einen bestimmten Fall

[11] VectorT<T> ist ein Template-Container ähnlich dem STL Vektor, der im Rahmen dieser Arbeit entwickelt wurde.

[12] Diese Template-Klasse wurde ebenfalls im Rahmen dieser Arbeit erstellt. Sie besitzt identische Funktionen wie die Klasse VectorT<T> mit dem Unterschied, dass die Vektordimension zu Übersetzungsphase bekannt ist.

[13] siehe dazu in Listing 5.2 in Zeile 4

*ein Abbruchkriterium liefert. Eine partielle Spezialisierung einer Funktion kann verwendet werden, um für einen bestimmten Fall während der Übersetzungsphase eine konkrete Implementierung anzugeben. In diesem Fall dient diese als Abbruchkriterium der Rekursion während der Instanziierunsphase: Falls der Schleifenindex auf die maximale Anzahl der Vektorelemente (*uiDimension*) hochgezählt ist, wird die Instanziierunsphase abgebrochen, indem die partiell spezialisierte Funktion anstatt der Funktion aus Zeile 1-8 aufgerufen wird.*

Es könnte hier noch eine Version 4 geben, bei der die innere for*-Schleife analog zur Vorgehensweise bei der Version 3 während der Übersetzungsphase vereinfacht werden könnte. Dies wird hier bewusst nicht aufgeführt, da dies lediglich eine Wiederholung wäre. Die drei verschiedenen Version werden hinsichtlich ihrer Laufzeit und den dabei benötigten Taktzyklen auf TP1 untersucht. Ferner wird die resultierende Programmgröße (in Bytes) und die benötigte Übersetzungszeit unter Verwendung des GNU Compilers (Version 4.5.1) bei der höchsten Optimierungsstufe (O3) beurteilt.*

[TP] Testplattform

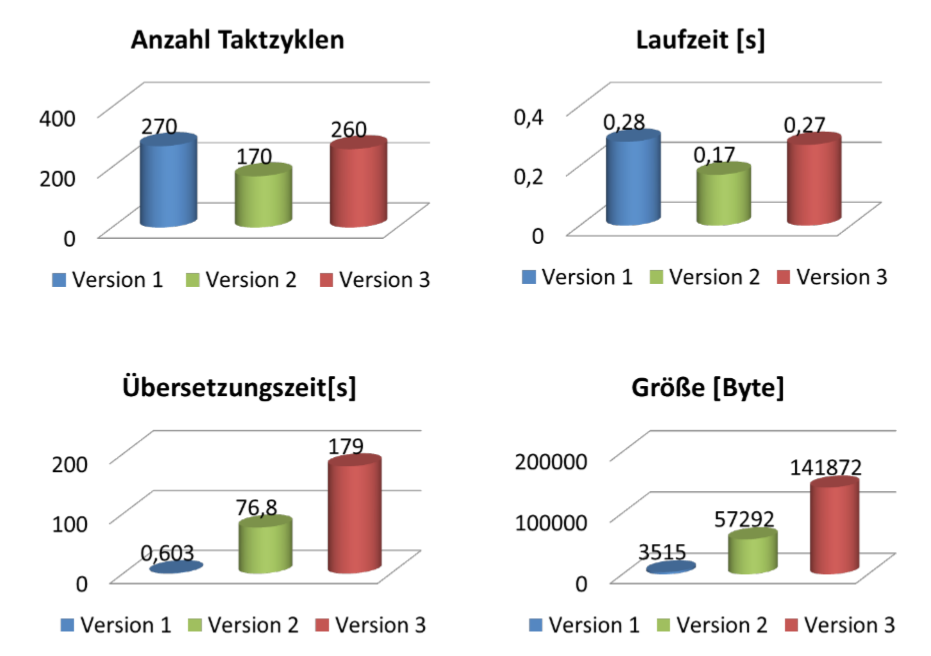


Abbildung 5.3: Gegenüberstellung der Laufzeit, Anzahl der Taktzyklen, Compilierungszeit und Größe von Template Metaprogrammen

Die Ergebnisse dieser Untersuchung sind in Abb. 5.3 dargestellt. Betrachtet man die Anzahl der eingeforderten Taktzyklen und die Laufzeit[14] *der jeweiligen Versionen, so benötigt die Version 2 am wenigsten Takte und folglich Laufzeit. Anscheinend wirkt sich das* loop-unrolling *der Version 3 negativ auf die Laufzeit aus. Ein weiteres Indiz hierfür zeigt ein Blick auf die Größe*[15] *der erzeugten ausführbaren Datei. Hierbei ist das erzeugte Programm um etwa das 2.5-fache der Version 2 und um das 40-fache der Version 1 größer. Auch die Übersetzungszeit des Programms der Version 3 schlägt mit nahezu 3 Minuten zu Buche, was etwas mehr als doppelt so lange dauert wie das Übersetzen der Version 2. Für diese Art der Aufgabenstellung scheint die Version 2 am besten geeignet, da hier ein guter Kompromiss zwischen Übersetzungszeit und Laufzeit gefunden werden konnte.*

[14]Diese wurde mit dem Linux *time* Kommando ermittelt. Hierbei wird auf die tatsächlich benötigte Zeitspanne Bezug genommen.

[15]Zur Bestimmung der jeweiligen Größe wurde das Linux size-Kommando verwendet.

```
1  template <typename T, unsigned int N> void Square(VectorTMP<T,N> &a)
2  {
3    VectorTMP<T,N> v(a);
4    for(unsigned int n=0; n < N; n++)
5    {
6      for(unsigned int i=0; i <= n; i++)
7      {
8        a[n] += v[i] * v[n-i];
9      }
10   }
11 }
```

Listing 5.2: Square Funktion (Version 2)

```
1      template <typename T, unsigned int n>
2      void OuterLoop(VectorTMP<T,uiDimension> &a
3             , const VectorTMP<T,uiDimension> &v)
4      {
5        for(unsigned int i=0; i <= n; i++)
6          a[n] += v[i] * v[n-i];
7        OuterLoop<T,n+1>(a,v);
8      }
9      template <> void OuterLoop<long double, uiDimension>
10     (        VectorTMP<long double,uiDimension>&
11     , const VectorTMP<long double,uiDimension>&){}
12
13     template <typename T, unsigned int N>
14     void Square(VectorTMP<T,N> &a)
15     {
16       VectorTMP<T,N> v(a);
17       OuterLoop<T,0>(a,v);
18     }
```

Listing 5.3: Square Funktion (Version3)

5.3.1.2 Vorlagen zur C++-Metaprogrammierung

Besonders das Abrollen von Schleifen während der Übersetzungsphase kann u.U. die Ausführungsgeschwindigkeit eines Programms erhöhen. Da TMP aufgrund ihrer andersartigen Syntax, verglichen mit Standard C++, schwerer lesbar sind, werden im Folgenden für gängige Anwendungsfälle wiederverwendbare Vorlagen ausgearbeitet. Hierzu wird zunächst mit einer *for*-Schleife aus Listing 5.4 begonnen. Diese Version der Schleife wird während der Laufzeitphase abgearbeitet, wobei die Index-Variable einen Wertebereich im Intervall $[0,10]$ überstreicht[16]. Das Pendant zu dieser Schleife ist in Listing 5.5 als Template Funktion angegeben. Eine weitere Variante dieser *for*-Schleife kann auch als Klassentemplate realisiert werden, diese findet man im Anhang auf Seite 227.

[16] Soll der Indexbereich auch negative Zahlen enthalten, ist es erforderlich, den Datentyp der Index-Variablen von *unsigned int* auf *int* zu ändern.

```
1 for(unsigned int uiIndex = 0; uiIndex <=10 ; uiIndex++)
2 { // Start
3 } // Ende
```

Listing 5.4: Laufzeit *for*-Schleife

```
1 template <unsigned int uiIndex> void For()
2 {
3     // [Index,...,1]
4     For<uiIndex-1>();
5     // [1,...,Index]
6 }
7 template <> void For<0>(){}
```

Listing 5.5: TMP *for*-Schleife

Die *for*-Schleife aus Listing 5.4 iteriert beginnend vom Wert 0 zum Wert 10, wohingegen die *for*-Schleife aus Listing 5.5 sowohl aufwärts ($0 \rightarrow 10$) als auch abwärts ($10 \leftarrow 0$) iteriert. Die Richtung hängt davon ab, an welcher Stelle der Wert der Index-Variable verwendet wird. Wird dieser vor dem rekursiven Aufruf verwendet (Zeile 3), dann erfolgt die Iteration abwärts. Andernfalls wird aufwärts gezählt, wenn der Wert der Index-Variable nach dem rekursiven Aufruf (Zeile 4) abgegriffen wird (Zeile 5). Wie schon im vorangegangenen Beispiel erwähnt, benötigt die Rekursion eine Bedingung zur Terminierung. In diesem Fall wird die rekursive Instanziierung gestoppt, sobald der Schleifenindex den Wert 0 erreicht. Da die Rekursion während der Übersetzungsphase durchgeführt wird, ist während der Schleife keine Schleife mehr vorhanden.
Im nächsten Schritt soll die *for*-Schleife zu einer zweifach geschachtelten Schleife erweitert werden und dafür ebenfalls ein Muster angegeben werden. Die erweiterte *for*-Schleife ist in Listing 5.6 angegeben. Hierbei ist zu beachten, dass die innere *for*-Schleife (Zeile 3) vom Index der äußeren Schleife (Zeile 1) abhängig ist.

```
1 for(unsigned int uiIndex1 = 0; uiIndex1 <=10 ; uiIndex1++)
2 { // Start
3   for(unsigned int uiIndex2 = 0; uiIndex2 <= uiIndex1; uiIndex2++)
4   { // Start
5   } // End
6 } // End
```

Listing 5.6: Laufzeit *for*-Schleife, zweifach geschachtelt

```
1 template <unsigned int uiIndex> void InnerFor()
2 {
3   InnerFor<uiIndex-1>();
4 }
5 template <> void InnerFor<0>(){}
6
7 template <unsigned int uiIndex> void For()
8 {
9   // [Index,...,1]
10  For<uiIndex-1>();
11  // [1,...,Index]
12  InnerFor<uiIndex>();
13 }
14 template <> void For<0>(){}
```

Listing 5.7: TMP *for*-Schleife, zweifach geschachtelt

Das Metaprogramm zur zweifach geschachtelten *for*-Schleife ist in Listing 5.7 dargestellt. Hierbei ist es nötig, für jede Schachtelungsebene eine eigene Funktion zu definieren. Diese ist in Zeile 1 bis 5 dargestellt und wird in Zeile 12 aufgerufen. Die bereits erwähnte Abhängigkeit der Index-Variablen wird hier durch einfaches Einsetzen des Wertes in den Aufruf der inneren Schleife erreicht (siehe Zeile 12). Auch hierfür gibt es eine weitere Lösungsvariante in Form von Klassentemplates, welche in Listing G.2 auf Seite 227 angegeben ist.
Eine n-fach verschachtelte *for*-Schleife erfordert somit n Funktionsdefinitionen mit eindeutigen Namenskonventionen. Aus diesem Grund empfiehlt es sich in der Praxis, die inneren Schleifen in einem eigenen C++-*namespace* zu kapseln. Dadurch können schon im Vorfeld diese möglichen Mehrdeutigkeiten vermieden werden. In diesem Abschnitt wurden lediglich Vorlagen für *for*-Schleifen erarbeitet. Andere Schleifentypen, wie beispielsweise *while*-Schleifen, wurden als Funktions- u. Klassentemplates in analoger Vorgehensweise entwickelt und sind in Abschnitt G.3 auf Seite 228 zu finden. Weitere Vorlagen zu *if-else* und *switch-case*-Kontrollstrukturen sind auf den Seiten 228 und 229 zu finden. Die hier erarbeiteten Vorlagen stellen ein Grundgerüst dar, mit dem ein herkömmliches C++-Programm in ein C++- Templatemetaprogramm umgewandelt werden kann.

5.3.1.3 Verwendung von TMP zur Lösung der homogenen Van-der-Pol-Gleichung

Im folgenden Beispiel soll die homogene Van-der-Pol-Gleichung[17] gelöst werden und dabei als erstes praktisches Beispiel dienen, die Eignung von Metaprogrammierungstechniken und deren Laufzeitverhalten für die Integration gewöhnlicher Differentialgleichen zu beurteilen.

Beispiel 5.3 für „Laufzeitvergleich zwischen TMP und herkömmlicher Programmierung“
Die Van-der-Pol'sche Gleichung:

$$\ddot{x} = -\eta(x^2 - 1)\dot{x} + x \qquad (5.3.1)$$

mit $\eta \in R$. Für $\eta = 0$ geht die Gleichung zum harmonischen Oszillator über. In den folgenden Simulationen wurden diese AW und Startbedingungen verwendet: $x = 1$, $\dot{x} = 0$ und

[AW] Anfangswerte

[17] siehe mehr dazu unter [Bronstein08] auf Seite 900.

$\eta = 1.8$. Ferner wurde die Bewegung des Oszillators über 1000 Integrationsschritte mit einer Schrittweite von $h = \frac{1}{10}$ berechnet, wobei der RK4 und RK10 - Integrator zur Lösung Verwendung fand. Das Verhalten der Van-der-Pol Gleichung für die hier definierten AW ist in Abb.5.4 dargestellt.

[RK4] Runge-Kutta 4.Ordnung

[RK10] Runge-Kutta 10.Ordnung

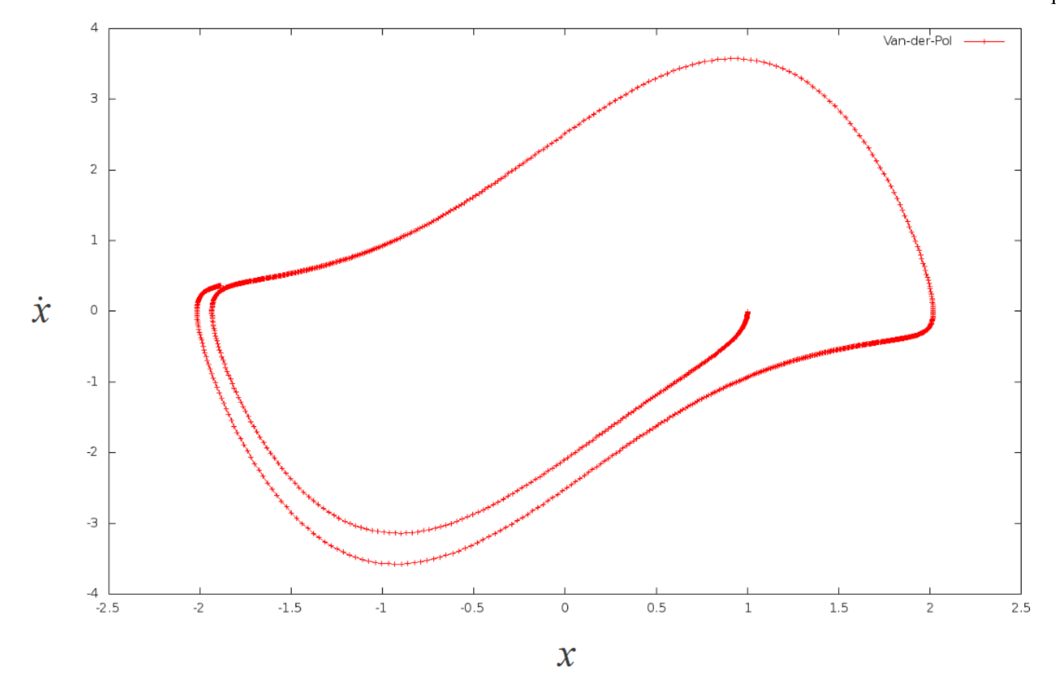


Abbildung 5.4: Lösung der Van-der-Pol'schen Gleichung

Um herauszufinden, welche Auswirkung TMP-Techniken auf das Laufzeitverhalten haben, wurden zwei verschiedene Versionen zur Lösung der Van-Der-Pol'schen-Gleichung implementiert. Die beiden Versionen unterscheiden sich nur anhand ihrer Basiskomponenten. Bei der ersten Version (ohne TMP) kommen Klassenobjekte, wie VectorT *zum Einsatz, welche beispielsweise ihre Dimension zur Laufzeit bereitstellen. Außerdem werden hierbei die einzelnen Integrationsschritte innerhalb einer* for*-Schleife durchgeführt. Bei der Version mit TMP wurde die benötigte* for*-Schleife mit Hilfe der erwähnten Methode während der Übersetzungszeit abgerollt. Außerdem basiert diese Version auf dem VectorTMP-Objekt, das seine Dimension während der Übersetzungsphase dem Compiler zur Verfügung stellt. Für die Simulationen kam TP1 mit g++-4.5*[18] *zum Einsatz. Um mögliche Einflüsse des Betriebssystems herauszumitteln, wurden die jeweiligen Messungen mehrfach*[19] *durchgeführt. Die Ergebnisse hierfür sind in Abb.5.5 dargestellt.*

[TP] Testplattform

[18] Es wurden folgende Optimierungsoptionen verwendet: *-O3,-march=nocona,-fno-strict- aliasing,-flto*

[19] sechs Mal pro Konfiguration

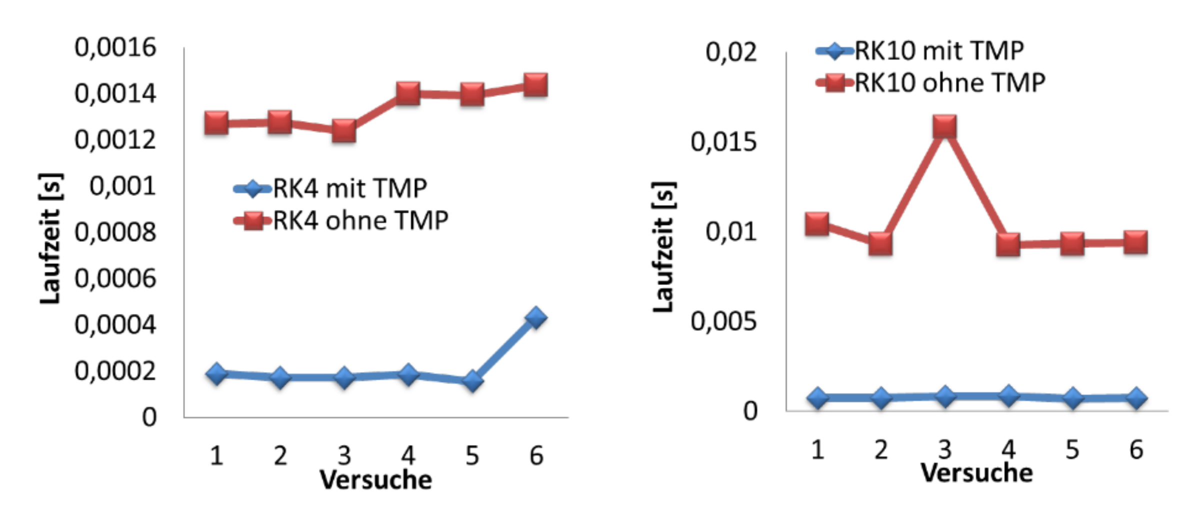


Abbildung 5.5: Ergebnisse aus dem Laufzeitvergleich zwischen TMP und herkömmlicher Programmierung

Dementsprechend wird schnell klar, dass die verwendete TMP-Technik die Laufzeit in allen durchgeführten Versuchen erheblich verkürzt. Wird aus den jeweils gemessenen Laufzeiten ein Mittelwert gebildet, so können die verschiedenen Lösungsvarianten gegenübergestellt werden. Die Auswertung ergibt, dass im Falle des RK4-Integrators die TMP-Variante um etwas mehr als das 6-fache und beim RK10-Integrator um das 14-fache schneller ist, als die Variante ohne TMP.

5.3.1.4 Anwendung von TMP auf die Berechnung von Satellitenbahnen

Im nächsten Beispiel wird untersucht, um welchen prozentualen Anteil die Laufzeit eines Simulationsprogramms zur Berechnung einer Satellitenbahn beschleunigt werden kann, falls bei der Programmierung Metaprogrammierungstechniken verwendet werden.

Beispiel 5.4 für „Simulation eines LAGEOS Orbits im anisotropen Gravitationsfeld der Erde“

[LAGEOS] LAser GEOdynamics Satellite

In diesem Beispiel soll die Bahn des LAGEOS[20] Satelliten numerisch integriert werden. Es wird ausschließlich die gravitative Anziehungskraft der Erde berücksichtigt und andere Kräfte, die auf den Satelliten wirken, werden vernachlässigt. Hierfür wird das EGM96[21]-Modell verwendet, mit dessen Hilfe die Anziehungskraft auf den Satelliten in jedem errechneten Bahnpunkt simuliert werden kann. Dabei ist bei jedem Aufruf der Kraftfunktion eine Kugelfunktionsentwicklung eines bestimmten Entwicklungsgrades l und einer bestimmten Ordnung m durchzuführen. Bei dieser Rechnung gilt: Je höher der Entwicklungsgrad, bzw. die Ordnung der Kugelfunktionsentwicklung, desto genauer kann die tatsächlich auf den Satelliten ausgeübte gravitative Anziehungskraft der Erde auf den Satelliten approximiert werden. Für diese Simulationen werden verschiedene Entwicklungsgrade im Intervall gewählt (mit $l, m \in [4, 50]$), um den zeitlichen Berechnungsaufwand in Abhängigkeit dieser Parameter sichtbar zu machen. Die Kugelfunktionsentwicklung wird programmiertechnisch mit for-Schleifen realisiert, was den Einsatz der TMP-Techniken begünstigt.

[EGM] Earth Gravity Modell

[20] Mehr Information zur LAGEOS-Mission ist auf der Webseite des Projekts zu finden: http://ilrs.gsfc.nasa.gov/satellite_missions/list_of_satellites/lag1_general.html

[21] Siehe mehr dazu unter [EGM96]

Ferner wurde die Bahnkonfiguration so gewählt, dass sie möglichst genau mit der tatsächlichen Bahn des LAGEOS-1 Satelliten übereinstimmt. Dieser benötigt im Mittel 225 Minuten pro Umlauf (6,4 Umläufe pro Tag) mit 5860 Kilometer Perigäumsdistanz. Für diese Simulation wird ein Integrationszeitraum von einer Woche gewählt, was 44,8 Umläufen entspricht. Zum Laufzeittest wurde die TP1 mit g++-4.5[22] *eingesetzt. Ferner wurde der* long double*-Datentyp der C-Standardbibliothek zur Berechnung ausgewählt. Um die Größe des übersetzten Programms zu verringern wurde das Hilfsprogramm* strip[23] *verwendet. Dieses Hilfsprogramm entfernt aus dem kompilierten Programm unnötige Symbole, wodurch die Größe reduziert wird. Zur Bestimmung der Größe des ausführbaren Programms wurde das Linux Programm* size[24] *verwendet. Dieses gibt sowohl die Anzahl der tatsächlich von der CPU benötigten Bytes als auch die Größe des statisch vorhanden Daten (ebenfalls in Byte) an.*

[TP] Testplattform

[CPU] Central Processing Unit

In Abb. 5.6 sind die gemessenen Laufzeiten in Abhängigkeit des Entwicklungsgrades ($l = m$) für die Berechnungsvarianten mit und ohne TMP-Techniken angegeben. Wie ersichtlich wird, benötigt die Berechnung mit TMP bei allen Simulationen deutlich weniger Laufzeit als die herkömmliche Methode ohne TMP. Die hierbei erzielte Verbesserung liegt, je nach Grad und Ordnung, im Bereich von 10 bis 40 Prozent. Hier soll noch auf den Anstieg der Programmgröße bei der Verwendung von TMP-Techniken hingewiesen werden (siehe Abb. 5.6 kleines Diagramm[25]*). Der Grund hierfür liegt am erzwungenen Abrollen der jeweiligen* for*-Schleifen und dem damit verbundenen Aufblähen des Quellcodes. Werden zu viele dieser Vereinfachungen durchgeführt, so steigt die Programmgröße u.U. derart stark an, dass sich dies negativ auf die Laufzeit auswirkt. Es ist somit die Balance zwischen rekursiver Instanziierung und Programmgröße zu halten, um ein Optimum an Ablaufgeschwindigkeit zu erzielen. Der Anstieg bei der Rechnung mit TMP-Techniken beim Entwicklungsgrad von 19 konnte auch durch mehrfaches Wiederholen der Simulation nicht nachgestellt werden und scheint somit durch das* Scheduling *des Betriebssystems verursacht worden zu sein.*

[22] Dabei wurden folgende Optimierungseinstellungen verwendet: *-ftemplate-depth-60580 -O3 -march=nocona*.

[23] siehe mehr dazu unter folgendem Weblink: http://linux.die.net/man/1/strip oder auf der Linux man-page.

[24] siehe mehr dazu unter folgendem Weblink: http://manpages.ubuntu.com/manpages/gutsy/man1/avr-size.1.html

[25] Hier wurden die tatsächlich von der CPU benötigten Bytes verwendet. Diese wurden mit dem Linux Programm *size* ermittelt.

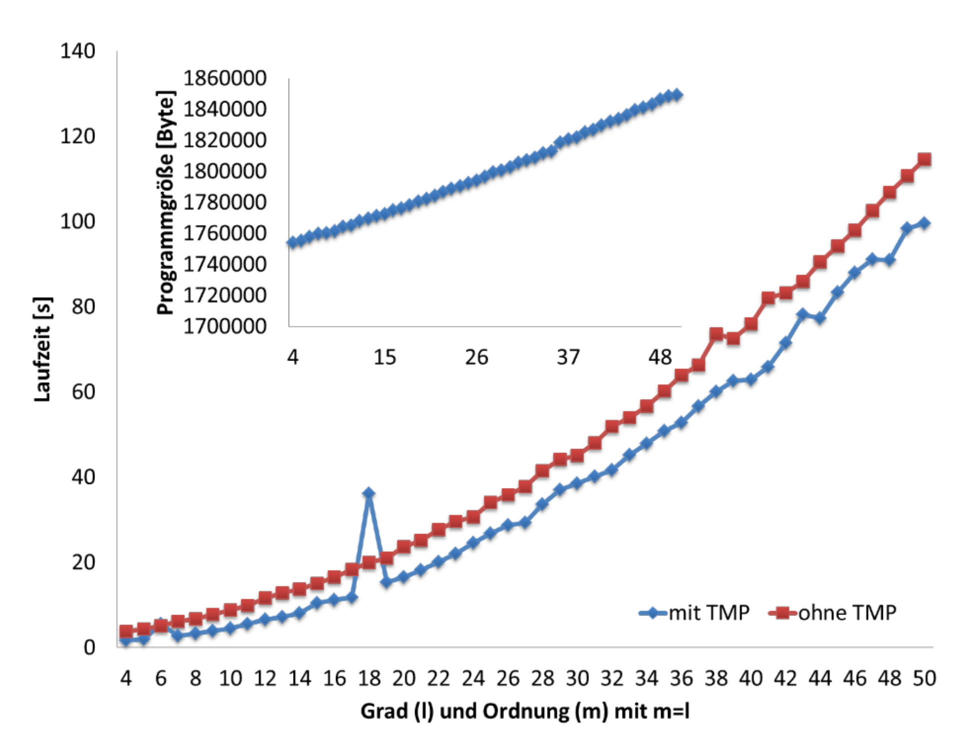


Abbildung 5.6: Ergebnisse der Laufzeitmessungen eines Programms zur Simulation eines LAGEOS Orbits in Abhängigkeit von Grad und Ordnung

Außerdem gibt es erhebliche Unterschiede in den benötigten Übersetzungszeiten. Die Variante mit TMP benötigte im Mittel 2.5 Minuten, bis das Programm kompiliert war, wohingegen die Variante ohne TMP im Mittel 1 Sekunde zum Übersetzen benötigte. Diese langen Übersetzungszeiten sind ein entscheidender Nachteil der TMP-Techniken und können beispielsweise verringert werden, indem auf einer leistungsfähigeren Maschine übersetzt wird. Zudem zeigten Experimente mit anderen Compilern, dass der clang-Compiler[26] *bei dieser Aufgabenstellung im Mittel bis zu einer Minute Übersetzungszeit einsparen konnte.*

5.3.1.5 Anwendung von TMP zur Berechnung trigonometrischer Funktionen

Eine weitere mögliche Anwendung für TMP-Techniken ist die Implementierung trigonometrischer Funktionen. Da trigonometrische Funktionen als Reihe dargestellt werden können und diese iterativ lösbar sind, kann hierfür auf die bereits erarbeiteten Methoden der TMP-Techniken zurückgegriffen werden. Somit können die Standardfunktionen, wie $sin, cos, tan, atan, ...$ als C++-Templatemetaprogramme implementiert werden. In folgender Berechnung wird das Prinzip am Beispiel der Sinus-Funktion (sin) aufgezeigt. Anschließend wird das Laufzeitverhalten der Standardbibliothek gegenübergestellt und beurteilt.

Beispiel 5.5 für „Implementierung der Sinus Funktion als C++-Templatemetaprogramm“

Die Sinus-Funktion soll in einer Reihe entwickelt und die ersten fünf Reihenterme, in Abhän-

[26]Hierbei wurde die Version 2.8 der Compilersuite verwendet. Siehe mehr dazu unter [CLangCompiler]

gigkeit des Parameters t, können wie folgt geschrieben werden[27]*:*

$$
\begin{aligned}
sin(t) &= t - \frac{1}{6}t^3 + \frac{1}{120}t^5 - \frac{1}{5040}t^7 + \frac{1}{362880}t^9 - ... \\
sin(t) &= t - \frac{t^3}{3!} + \frac{t^5}{5!} - \frac{t^7}{7!} + \frac{t^9}{9!} - ...
\end{aligned}
\tag{5.3.2}
$$

Soll nun ein Funktionswert an der Stelle t mit Hilfe eines Templatemetaprogramms berechnet werden, so sind hierfür Hilfsfunktionen zur Berechnung der Potenz und Fakultät erforderlich. Erstere berechnet aus einer gegebenen Gleitkommazahl (x) und einer angegebenen Hochzahl (i) die Potenz (x^i). Dies ist ebenfalls eine TMP-Funktion, worin die erforderliche Iteration während der Übersetzungsphase durchgeführt wird. Lediglich die Berechnung der Gleitkommazahlen werden hier während der Laufzeitphase durchgeführt. Die zweite Hilfsfunktion berechnet die Fakultät einer gegebenen Integerzahl. Diese wird vollständig während der Übersetzungsphase berechnet, sodass dadurch keine Laufzeit in Anspruch genommen wird. Die beiden Hilfsprogramme sind in den Listings G.9 und G.10 auf Seite 230 angegeben. Das Programm zur Berechnung des Sinus ist in Listing 5.8 angegeben. Dies soll im Folgenden detailliert erklärt werden:

- ***Zeile 1**:*
  Enthält die Funktionsdefinition mit zwei Template-Parametern. Hierbei ist T ein Platzhalter für den Typ der Gleitkommazahl und N ein Platzhalter für die Ordnung der Reihenentwicklung. Der Funktionsparameter x ist ein Platzhalter für den Abszissenwert. Soll beispielsweise der Sinus an der Stelle $x = 0.1$ unter Verwendung des $T = double$-Datentyps mit einer Ordnung $N = 20$ entwickelt werden, so würde der Aufruf wie folgt lauten: Sin<double,20>(0.1);

- ***Zeilen 3 bis 6**:*
  In dieser und den weiteren Zeilen wird der Bedingungsoperator $?$ verwendet[28]*. Wie in Gleichung 5.3.2 dargestellt, enthält die Reihenentwicklung eines Sinus ausschließlich ungerade Indices. Um dies zu gewährleisten, wird in der ersten Bedingung ermittelt, ob die Variable N einen geraden oder ungeraden Wert besitzt. Ist der Wert gerade, so wird der Index um eins reduziert und die Funktion wird erneut rekursiv aufgerufen. Falls der Wert ungerade ist, wird ebenfalls die Funktion rekursiv aufgerufen (Zeile 4). Hierbei wird der Index bei jedem Aufruf der Rekursion um zwei reduziert, solange bis die Rekursion terminiert (Zeile 10). In den Zeilen 5 und 6 werden die bereits erwähnten Hilfsfunktionen, mit dem jeweiligen Wert der Variablen N, aufgerufen, wobei vorher in Abhängigkeit des Wertes von N das Vorzeichen gesetzt wird.*

- ***Zeilen 8 bis 12**:*
  In den Zeilen 8 bis 10 ist ein Präprozessor-Makro definiert. Dieses wird in Zeile 12 aufgerufen und stellt das Kriterium der rekursiven Schleife dar (Zeile 10). In Zeile 9 wird zudem ein Sonderfall behandelt: Falls die Reihenentwicklung des Sinus mit der Ordnung 1 aufgerufen wird, dann wird der Abszissenwert zurückgegeben.

---

[27] siehe mehr dazu [Bronstein08]

[28] Dieser Operator entstammt der Programmiersprache C und kann als verkürzte Schreibweise einer *if-else*-Bedingung angesehen werden. Dessen Funktionsweise ist in [Kernighan90] in Abschnitt 2.11 erklärt.

```
1  template<typename T,unsigned int N> inline T Sin(const T &x)
2  {
3    return (!(N%2)) ? Sin<T,(!(N%2))?N-1:0>(x)
4                    : Sin<T,(N%2)?N-2:0>(x) +
5                      ((((N-1)/2)%2)?-1:1)*Pow<(N%2)?N:0,T>(x)
6                                          /Fact<(N%2)?N:0,T>();
7  }
8  #define TERMINATE_SIN_LOOP(TYPE)\
9  template<> inline TYPE Sin<TYPE,1>(const TYPE &x ){return x  ;}\
10 template<> inline TYPE Sin<TYPE,0>(const TYPE &  ){return 0.0;}
11
12 TERMINATE_SIN_LOOP(double)
```

Listing 5.8: TMP Sinus

Nach Vorgabe des oben durchgeführten Beispiels wurden einige trigonometrische Funktionen der C-Standardbibliothek als Templatemetaprogramme implementiert. Um deren Leistungsfähigkeit hinsichtlich numerischer Stabilität und Ablaufgeschwindigkeit beurteilen zu können, werden diese einem Leistungstest (*Benchmark*) unterzogen. Hierbei wird das Programm *fbench*[29] verwendet. Dieses wurde ursprünglich zur Leistungsmessung und Verifikation[30] trigonometrischer Funktionen der C-Standardbibliothek verwendet und für diesen Zweck umgeschrieben, sodass auch C++ Programme überprüft werden können. Damit ist es nun möglich, die trigonometrischen Funktionen, welche auf der TMP-Technik basieren, auf ihre Korrektheit hin zu überprüfen. Außerdem kann ein Laufzeitvergleich mit den Routinen der C-Standardbibliothek durchgeführt werden[31]. Die Ergebnisse der Laufzeitmessungen sind in Abb 5.7 dargestellt. Um mögliche Beeinflussungen durch des Betriebssystem herauszumitteln, wurde pro Konfiguration jeder Benchmark fünfmal durchgeführt (auf TP1). Außerdem wurden zwei verschiedene Compiler verwendet: der GNU C++-Compiler (g++[32]) und der Clang-Compiler (clang++[33]).

[TP] Testplattform

[29] Dies ist ein Programm zur Verifikation und Leistungsmessung trigonometrischer C-Funktionen. Siehe mehr dazu unter [fbench].

[30] Hierfür wird ein *ray tracing* - Algorithmus verwendet. Dieser basiert fast ausschließlich auf trigonometrischen Transformationen. Mehr Informationen zum *ray tracing*-Algorithmen können in [Shirley07] nachgeschlagen werden.

[31] Das Benchmarkprogramm wurde dabei wie folgt aufgerufen: *time ./fbench 10000000*

[32] Version 4.5.1

[33] Version 2.8

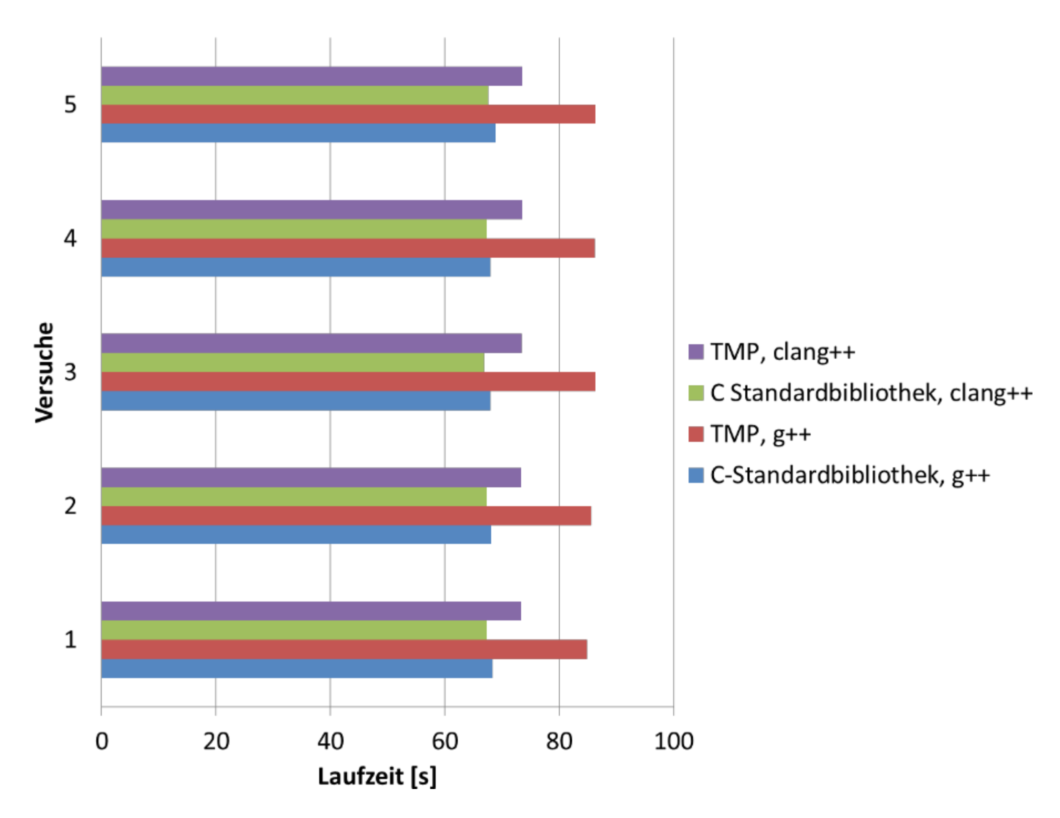


Abbildung 5.7: Resultate der Laufzeitmessungen beim fbench-Programm

Aus den Laufzeitmessungen wird ersichtlich, dass die erzielten Laufzeiten sehr stark vom verwendeten Compiler und dessen Optimierung während des Übersetzungsvorgangs abhängig sind. Bei allen fünf Versuchen, sowohl bei TMP-Funktionen, als auch bei den Funktionen der C-Standardbibliothek, erzeugte der Clang-Compiler effizientere Programme. Außerdem war bei keinem der Versuche die TMP-Variante schneller als die der Standardbibliothek und stellt somit keinen effizienteren Ersatz zu den Standardbibliotheksfunktionen dar. Dennoch konnte gezeigt werden, dass die TMP-Technik prinzipiell geeignet ist, derartige Problemstellungen zu lösen. Außerdem deutet der auffällige Laufzeitunterschied zwischen *g++* und *clang++* darauf hin, dass durch weitere Verbesserung der Compilersuiten und insbesondere der darin verwendeten Optimierungsroutinen ein C++-Templatemetaprogramm die Geschwindigkeit einer Routine der Standardbibliothek erreichen kann. Darüber hinaus wäre es interessant, welche Laufzeitunterschiede sich bei sonst identischer Hardware, aber unterschiedlicher Prozessorarchitektur (32/64-Bit), ergeben würden.

5.3.1.6 Bewertung von TMP-Techniken

Soll bei einem Programm TMP zum Einsatz kommen, dann sollten folgende Gesichtspunkte bedacht werden:

- **Verhältnis zwischen Übersetzungszeit und Ausführungszeit**
  Da Templatemetaprogramme während der Instanziierunsphase von Templates verarbeitet werden, steigt der zeitliche Aufwand u.U. enorm. Dabei ist jedoch

nicht immer sichergestellt, dass das erstellte Programm an Ausführungsgeschwindigkeit zulegt, wie einige der behandelten Beispiele[34] gezeigt haben. Folglich ist es wichtig, die Balance zwischen Übersetzungs- u. Ausführungszeit zu finden, da zu exzessives Anwenden dieser Technik zu nicht praktikablen Übersetzungszeiten führt.

- **Anwachsen der Programmgröße**
  Durch massiven Einsatz rekursiver Vereinfachung von *for*-Schleifen kann die Programmgröße derart ansteigen, dass sich dies negativ auf die Ausführungsgeschwindigkeit auswirkt. Es ist somit ein Mittelweg zwischen rekursiver Template-Instanziierung und Programmgröße anzustreben, um ein Optimum an Ausführungsgeschwindigkeit zu erreichen.

- **Lesbarkeit und Wartbarkeit**
  Die Syntax von Templatemetaprogrammen ist durch ihre rekursive Struktur anfangs gewöhnungsbedürftig und erfordert einiges an Erfahrung. Dies wirkt sich nachteilig auf die Wartbarkeit des erstellten Quellcodes aus.

- **Portabilität**
  Da ein Templatemetaprogramm ausschließlich vom Compiler verarbeitet wird, ist es folglich auch von dessen interner Implementierung abhängig. Somit ist weder gewährleistet, dass jeder Compiler bzw. Version des Compilers ein Templatemetaprogramm verarbeiten kann. Hinzu kommen noch plattformbezogene Eigenheiten der jeweiligen Compiler, welche eine Portierung zusätzlich erschweren können.

- **Auffinden von Fehlern**
  Das Auffinden von Fehlern während der Entwicklungsphase wird hier besonders erschwert. Dabei kann es beispielsweise bei einem syntaktisch falschen Programm vorkommen, dass der Compiler hunderte von Seiten Fehlermeldungen produziert, ohne auf das tatsächliche Problem hinzuweisen. Ferner gibt es keine Programme zur Analyse von Templatemetaprogrammen. Die Entwicklung derartiger Programme wird zusätzlich erschwert, da es nicht möglich ist, an bestimmten Programmstellen Zwischenwerte auszugeben.

5.3.1.7 Fazit und weiterführende Literatur

Die Verwendung von C++-Templatemetaprogrammierung kann zu signifikanten Geschwindigkeitsvorteilen führen, es darf jedoch die benötigte Einarbeitungszeit und der nötige Aufwand während der Entwicklungsphase nicht unterschätzt werden. Des Weiteren kann das Buch von [Stroustrup04] empfohlen werden. Darin wird das Konzept der Templatemetaprogrammierung vorgestellt und auf die Funktionsweise der MPL[35] eingegangen. Ferner bietet das Buch von [Vandevoorde08] eine gute Hilfestellung zur Einarbeitung in dieses Thema. Darin wird u.a. ein allgemeiner Überblick zur Templateprogrammierung, als auch zur Templatemetaprogrammierung geboten. Ergänzend soll

[MPL] Boost Metaprogramming Library

[34] siehe mehr dazu auf Seite 107

[35] siehe mehr dazu unter [MPL]

hier noch auf das *blitz++*[36]-Projekt hingewiesen werden. Hierbei handelt es sich um eine C++-Bibliothek, in der Metaprogrammierungstechniken eingesetzt wurden, um auf Geschwindigkeit hin optimierte mathematische Operationen zu realisieren. Dabei werden so genannte *expression*-Templates eingesetzt. Dies ist eine Metaprogrammierungstechnik, bei der durch massive Überladung von Operatoren die Vereinfachung von mathematischen Ausdrücken in die Übersetzungsphase verlagert, d.h. dem Compiler überlassen wird. Außerdem ist das Buch von [Alexandrescu01] erwähnenswert. Dort werden *Design-Pattern* zur Lösung von Standardproblemen der Informatik angegeben. Diese werden mit Hilfe von TMP-Techniken gelöst, detailliert erläutert und mit prägnanten Beispielen untermauert.

5.3.2 Konzepte der parallelen Programmierung

Der derzeitige Trend, immer mehr Prozessorkerne in einen Computerchip zu integrieren kann u.U. eine erhebliche Leistungsverbesserung für bestimmte Anwendungsfälle bedeuten. Ganz im Gegenteil zu früheren Chipverbesserungen, wie die Erhöhung des Prozessortaktes, hat diese Änderung einen Einfluss auf die Softwareentwicklung und deren Methoden. Hierbei kann prinzipiell von zwei verschiedenen Voraussetzungen ausgegangen werden. Wird der Speicher, welcher während einer parallelen Abarbeitungssequenz benötigt wird, von mehreren parallelen Ablauffäden geteilt, so spricht man i.A. von einer *shared memory*-Architektur. Besitzt andererseits jeder parallele Ablauffaden seinen eigenen Speicherbereich, welcher unabhängig vom anderen Speicher ist, so wird dies als *distributed memory*-Architektur bezeichnet. Dies ist beispielsweise der Fall, falls mehrere Computer mit Hilfe eines Computernetzwerks zu einem Rechnerverbund zusammengeschaltet werden, um eine bestimmte Aufgabe zu lösen. Diese beiden Architekturen sind in Abb. 5.8 illustriert.

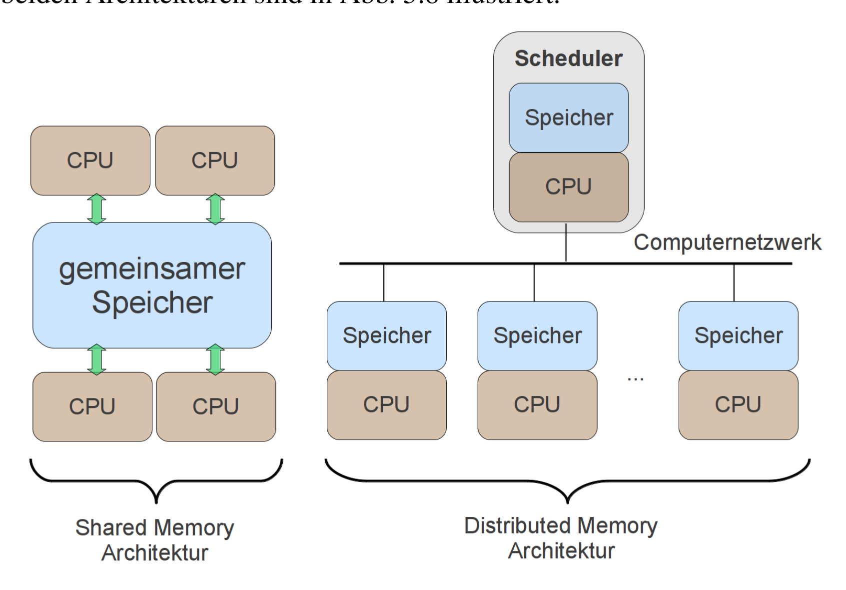


Abbildung 5.8: Konzepte paralleler Architekturen

Auf der linken Seite ist schematisch eine *shared memory*-Architektur abgebildet, bei der sich mehrere CPUs einen gemeinsamen Speicher teilen. Auf der rechten Seite

[CPU] Central Processing Unit

[36] siehe mehr dazu unter [blitz++]

ist eine *distributed memory*-Architektur dargestellt. Hiermit können parallele Aufgaben über ein Netzwerk verteilt werden. Demzufolge besitzt jeder Knoten eine oder mehrere CPUs und einen abgeschlossenen Speicher. Ein *Scheduler* ist dafür verantwortlich, die Aufgaben an die jeweiligen Knoten zu verteilen. Aus diesem Grund werden derartige Architekturen in der Literatur oft als verteilte Systeme[37] bezeichnet.

5.3.2.1 *Shared Memory* Parallelisierung [SMP] mit OpenMP

[OpenMP] Open Multi-Processing

OpenMP ist eine Programmierschnittstelle, mit welcher ein Programm oder bestimmte Programmteile nach dem *shared memory*-Prinzip parallelisiert werden können. Um einen bestehenden Programmteil zu parallelisieren, wird zu Beginn eine Pragma Direktive[38] geschrieben. Somit ist dem Compiler bekannt gegeben, dass an dieser Stelle ein Bereich beginnt, der parallelisiert werden kann. Das Arbeitsprinzip von OpenMP ist in Abb. 5.9 dargestellt.

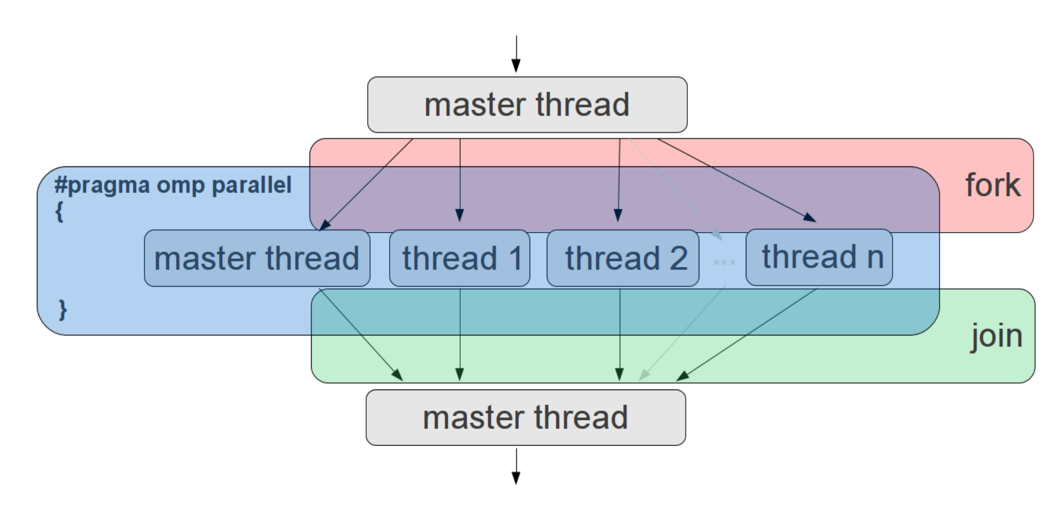


Abbildung 5.9: Das Arbeitsprinzip von OpenMP

Die parallele Ausführung eines Programmteils mit OpenMP wird mit Hilfe von *threads*[39] realisiert. Zu Beginn eines jeden OpenMP Programms wird ein *thread*, der *master thread*, aktiviert und kann als Hauptablauffaden betrachtet werden. Wird nun während der Ausführung eine Pragma Direktive verwendet (z.B. `#pragma omp prallel`), so erzwingt dies eine Aufteilung in mehrere parallele Ablauffäden (*threads*). Dieser Vorgang wird i.A. als *fork* bezeichnet. Diese parallelen *threads* teilen sich die nötige Arbeit, was u.U. mit einem erheblichen Leistungszuwachs verbunden sein kann. Haben alle *threads* ihre Arbeit erledigt, dann werden die parallelen *threads* terminiert und vorher die Ergebnisse vom *master thread* eingesammelt. Diese Vorgang wird als *join* bezeichnet.

[37]Mehr zum Thema der verteilten Systeme und den damit verbundenen Algorithmen findet man in [Ajay08]

[38]Eine Pragma-Direktive weist den Compiler an, die durch die Pragma Direktive bezeichnete und meist von der Implementierung des Compilers abhängige Operation durchzuführen. Ist diese Direktive dem Compiler nicht bekannt, dann wird diese ignoriert. Pragma-Direktiven sind im ANSI-C-Standard definiert. Dieser kann unter http://www.open-std.org/jtc1/sc22/wg14/www/standards eingesehen werden.

[39]*Threads* können als parallele Ablauffäden innerhalb eines Prozesses aufgefasst werden.

Beispiel 5.6 „Simulation eines LAGEOS Orbits im anisotropen Gravitationsfeld der Erde mit Hilfe von OpenMP“
Um die Einsatzfähigkeit von OpenMP für die Integration einer Satellitenbahn zu demonstrieren, soll das Beispiel aus Seite 114 wieder aufgegriffen werden. Hierfür wurden bei der Berechnung der Beschleunigung - im Quellcode der Kraftfunktion - welche durch das Gravitationsfeld der Erde auf den Satelliten ausgeübt wird, OpenMP-Direktiven angebracht. Dabei wurden die Berechnungen in unabhängige Teile zerlegt, sodass diese Teile parallel verarbeitet werden können. Dies wurde für beide Varianten des vorausgegangenen Beispiels, mit und ohne TMP-Technik, durchgeführt. Alle anderen Randbedingungen wurden belassen, um möglichst vergleichbare Resultate zu erzielen. Die Ergebnisse der Laufzeitmessungen sind in Abb. 5.10 dargestellt. Bei höheren Entwicklungsgraden ist die Variante mit OpenMP und diejenigen, bei denen TMP-Techniken zum Einsatz kamen am schnellsten. Bei niedrigeren Entwicklungsgraden weist die Variante mit OpenMP etwas langsamere Laufzeiten auf. Dies liegt daran, dass hier, sobald eine parallele Abarbeitung nach dem for-join*-Prinzip beginnt, zusätzlich Laufzeit verbraucht wird. Dies zeigt, dass sich der Einsatz von OpenMP erst ab einem gewissen Aufwand lohnt. Vergleicht man die beiden Varianten, ohne Einsatz von TMP und ohne OpenMP mit dem Einsatz von TMP und mit OpenMP, so kann durch letztere Variante im Mittel 33,6% Laufzeit eingespart werden, was durchaus lohnenswert erscheint.*

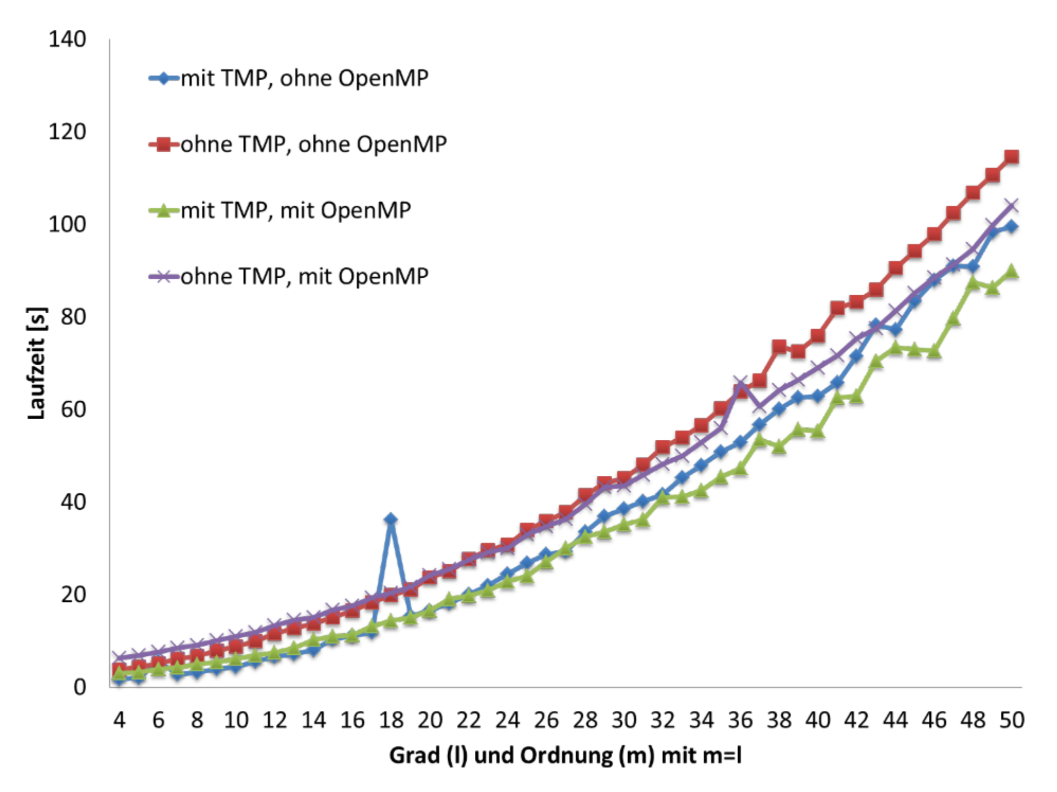


Abbildung 5.10: Ergebnisse der Laufzeitmessungen bei Verwendung von TMP-Techniken und OpenMP

5.3.2.2 Fazit und weiterführende Literatur

In der durchgeführten Beispielsimulation konnte die Methode der Parallelisierung mit OpenMP zwar überzeugen, es ist aber dennoch Vorsicht geboten, da bei der Programmierung darauf geachtet werden muss, so genannte *race conditions*[40] zu vermeiden. Aus diesem Grund ist es empfehlenswert, das Programm schon während der Entwicklungsphase ohne OpenMP zu verifizieren. Diese und andere Fallstricke im Bereich der parallelen Programmierung können schon während der Entwicklungsphase erkannt werden. Hierzu können Hilfsprogramme wie der *Intel®Inspector XE*[41], als auch Cppcheck[42] oder Polyspace[43] verwendet werden. Die hier gezeigte Methode der Parallelverarbeitung ist nicht die einzige. Es gibt weitere Möglichkeiten, ein serielles Programm zur Parallelverarbeitung umzuschreiben[44]. Eine weitere, derzeit häufig genutzte Variante ist OpenMPI[45]. Hierbei handelt es sich um eine Infrastruktur zu Realisierung eines verteilten Systems (*distributed memory*-Architektur). Eine MPI-Anwendung ist i.d.R. in mehrere Prozesse aufgeteilt, welche dann miteinander kommunizieren. Diese hat den Vorteil, dass sie auch über Rechnergrenzen hinweg realisiert werden kann. Dieser Ansatz wurde hier nicht weiter verfolgt, da sonst der Rahmen dieser Arbeit gesprengt werden würde.

[OpenMPI] Open Message Passing Interface

Zur Programmierung mit OpenMP wird auf die Bücher von [Hoffmann08], [Chapman07], [Quinn04] und [Hughes03] hingewiesen. Darüber hinaus gibt es einige Ansätze, die Parallelisierung weitestgehend in die Übersetzungsphase zu verlagern. Ein nennenswertes Projekt ist der ISPC-Compiler[46], welcher Programme in C-ähnlicher Syntax übersetzen kann. Hierbei wird dem Programmierer die Verantwortung abgenommen, ein Programm an bestimmten Stellen gezielt zu parallelisieren. Ein entscheidender Nachteil ist die bereits erwähnte C-ähnliche Syntax, bei der gewisse Sprachmittel wie beispielsweise *switch-case*-Bedingungen gänzlich fehlen. Dennoch ist dieser Ansatz zukunftsweisend und sollte von den Entwicklern von Compilern berücksichtigt werden. Hierbei wäre es wünschenswert, wenn derartige Funktionalitäten beispielsweise in den GNU-Compiler integriert würden. Somit könnten ältere Programme, die ursprünglich für serielle Verarbeitung programmiert wurden, auch von Mehrkernprozessoren profitieren. Außerdem zu erwähnen ist der relativ neue Standard für parallele Programmierung OpenACC[47]. Dieser stellt eine Weiterentwicklung des OpenMP-

[ISPC] Intel®SPMD Programm Compiler

[40] Dies sind Programmkonstrukte, in denen das Ergebnis einer Operation vom zeitlichen Verhalten bestimmter Einzeloperationen abhängig ist.

[41] siehe mehr dazu unter http://software.intel.com/en-us/articles/intelinspectorxe/

[42] Dies ist ein statisches Quellcodeanalyseprogramm mit dem u.a. *deadlocks* identifiziert werden können. Mehr Informationen können unter http://cppcheck.sourceforge.net/ nachgeschlagen werden.

[43] Dies ist ein proprietäres und sehr umfangreiches Programm, welches neben vielen anderen Quellcodeanalysen auch *race-conditions* im Quelltext aufspüren kann.

[44] siehe mehr zu alternativen Ansätzen der Parallelverarbeitung mit C++ unter [Rauber08], ab Seite 145.

[45] Der OpenMPI Standard kann unter folgender Webadresse eingesehen werden http://www.open-mpi.org/doc/

[46] siehe mehr dazu unter [ISPC-Compiler]

[47] siehe mehr dazu auf der Webseite des Projekts: http://www.openacc-standard.org/home, Download 15.11.2011

Standards für PGI[48], CAPS[49] und Cray CCE[50] Compilersystemen dar, welcher kompatibel mit den gängigen CPU/GPU-Standards sein soll. Hierzu werden während der Übersetzungsphase durch den Compiler Programmstellen identifiziert, welche parallelisiert werden können. Diese Programmstellen werden anschließend dem Programmierer bekannt gegeben. Daraufhin kann der Programmierer Schritt für Schritt einer automatischen Parallelisierung zustimmen oder diese an bestimmten Stellen verbieten. Ob der jeweilige Programmteil auf der Grafikkarte oder der CPU ausgeführt wird, entscheidet in diesem Fall der OpenACC-Compiler. Erste Umsetzungen dieses Standards werden bis Mitte des Jahres 2012 erwartet.

[GPU] Graphics Processing Unit

5.4 Optimierung während der Übersetzungsphase

Dieser Abschnitt ist der Übersetzungsphase gewidmet, in der Programme mit Hilfe eines Compilers in die Maschinensprache übersetzt werden. Bei der Bedienung dieser Werkzeuge gibt es viele Möglichkeiten, die Geschwindigkeit des zu erzeugenden Programms zu beeinflussen. Vorweg sei auf die Artikel von [Novillo05-1] und [Novillo05-2] hingewiesen, die einen guten Überblick zu Optimierungsoptionen und zur Geschwindigkeitsanalyse bieten.

5.4.1 Die GNU Compiler Collection [GCC]

Die GNU Compiler Collection[51] ist eine Sammlung von Compilern für die Programmiersprachen C, C++, Java, Objective-C, Fortran, Ada und Go, wobei in dieser Arbeit lediglich auf die Compiler der Sprachen C (gcc) und C++ (g++) Bezug genommen wird. Diese Sammlung unterstützt derzeit (Juni 2011) bis zu 60 verschiedene Hardwareplattformen[52].

5.4.2 Die Wahl der optimalen Compileroption

Es ist keine leichte Aufgabe, die richtigen Einstellungen für einen Compiler zu finden, sodass dieser das Programm bestmöglich optimiert. Hierzu wird von den Compilerherstellern[53] und in der Literatur[54] für die jeweilige Version umfangreiches Informationsmaterial zur Verfügung gestellt. Entscheidend hierfür ist die richtige Wahl der jeweils benötigten Einstellungen. Bei geeigneter Optimierungseinstellung kann in bestimmten Fällen zwischen 20-40% Laufzeit eingespart[55] werden. Somit ist es durchaus lohnenswert, diesen Bereich genauer zu untersuchen.

[48] Mehr Informationen zu dieser Compilersuite können auf der Webseite des Projekts nachgeschlagen werden: http://www.pgroup.com/products/pgiworkstation.htm, Download 15.11.2011

[49] Nähere Informationen können auf der Webseite des Compilerherstellers nachgelesen werden: http://www.caps-entreprise.com/index.php, Download 15.11.2011

[50] siehe mehr dazu unter http://user.cscs.ch/nc/news/index.html, Download 15.11.2011

[51] siehe mehr dazu unter [GNUCompiler], [Gough04] und [VonHagen08]

[52] siehe mehr dazu unter http://gcc.gnu.org/install/specific.html, Download 22.06.2011

[53] siehe mehr dazu unter [GNUCompiler]

[54] siehe mehr dazu unter [VonHagen08], ab Seite 110

[55] Information aus [Bulka99], Seite 144

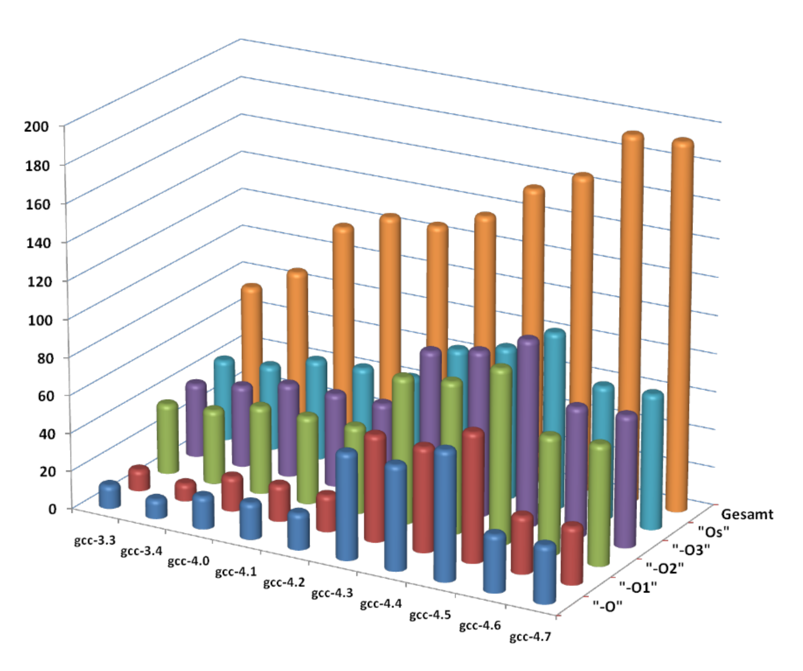


Abbildung 5.11: Anzahl der aktivierten Optimierungsoptionen bei vordefinierten Optimierungsstufen des GNU-C Compilers

Wie aus Abb. 5.11 ersichtlich wird, wurde im Laufe der Zeit eine zunehmende Anzahl verschiedener Optimierungsoptionen eingeführt, mit denen das zu übersetzende Programm auf verschiedene Art und Weise optimiert werden kann. Hatte die Version 3.3 des GNU-C Compilers noch 74 Optionen, mit denen die Optimierung seitens des Compilers beeinflusst werden konnte, so besitzt die aktuelle[56] Version des Compilers bereits 192 verschiedene Optionen[57]. Einige dieser Optionen führen zu schnellen Programmen, wenngleich aber die Größe des Programms u.U. zunehmen kann. Es gibt auch andere Optionen, welche die Größe des Programms verringern, dadurch jedoch nicht zwingend die Ausführungsgeschwindigkeit erhöhen. Um dem Benutzer u.a. die Nutzung des Werkzeugs zu vereinfachen, wurden von den Entwicklern verschiedene Optimierungsstufen eingeführt ($O1, O2, O3, Os$). Wird eine dieser Optimierungsstufen ausgewählt, so wird eine bestimmte vordefinierte Untermenge an Optimierungsoptionen aktiviert. Welche Optimierungsoptionen der jeweiligen Optimierungsstufe zugeordnet sind, hängt von der Version des Compilers ab. Einer bestimmten Optimierungsstufe ist jeweils ein Zweck zugeordnet[58]:

- $O1$
  Hierbei wird versucht, die Übersetzungszeit so kurz wie möglich zu halten und dennoch das Programm zu optimieren.

- $O2$
  Bei dieser Stufe erhöht sich die Übersetzungszeit und es wird versucht, das Programm besser zu optimieren, sodass kürzere Ausführungszeiten erreicht werden können.

[56]gcc-4.6, Juni 2011
[57]Die genaue Anzahl der jeweils aktivierten Optionen ist in Tabelle F.1 auf Seite 217 eingetragen.
[58]siehe mehr dazu unter [GNUCompiler]

- $O3$
  Dies ist die stärkste Optimierungsstufe. Hierbei werden i.d.R. die meisten Optimierungsoptionen aktiviert, was zu längeren Übersetzungszeiten als bei den niederen Stufen führt. Ziel dieser Stufe ist es, das Programm noch stärker zu optimieren, um noch höhere Ausführungsgeschwindigkeiten zu erreichen.

- Os
  Dabei handelt es sich um eine Ansammlung von Optimierungsoptionen, welche das Programm auf minimale Größe optimieren soll. Diese, auf minimale Größe optimierten Programme, finden oft auf Plattformen im *embedded*-Bereich ihre Anwendung.

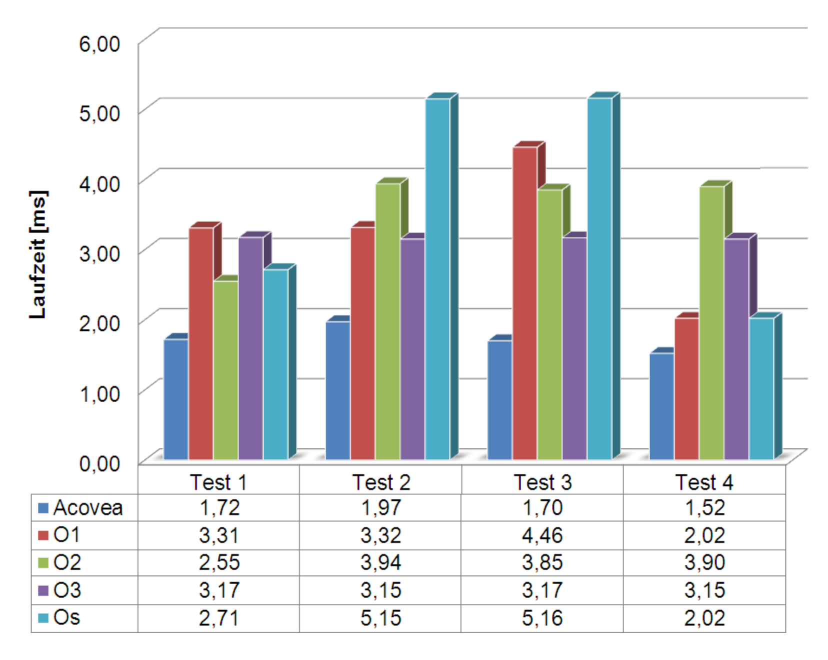
Außerdem wird [Aho95-2] (ab Seite 715) empfohlen, dort sind Informationen zur Codeoptimierung enthalten. Außerdem wird beispielhaft an praktischen Aufgabenstellungen gezeigt, wie eine Leistungsverbesserung für bestimmte Codefragmente erzielt werden kann. Ferner wird ein Einblick in Optimierungstechniken, wie sie in heutigen Compilern vorzufinden sind, geboten.
Soll nun eine Kombination aus Compileroptionen gefunden werden, die das Programm besser optimieren als die standardmäßig vorgegebenen, so kann das Problem kaum durch Ausprobieren bewältigt werden. Folgendes Beispiel verdeutlicht dies:

Beispiel 5.7 „Abschätzung des Aufwands zum Finden der besten Kombination von Compilerflags für ein Programm“
Falls beispielsweise 50 verschiedene Compilerflags vorhanden sind und diese sich unter Umständen wechselseitig beeinflussen, dann ergeben sich daraus 2^{50} Kombinationsmöglichkeiten. Dabei müsste für jede Kombination ein Programm übersetzt und dessen Laufzeitverhalten bewertet werden. Würde dies im Durchschnitt eine Sekunde pro Kombination beanspruchen, so würde die gesamte Suche 2^{50} Sekunden dauern. Damit würde dieser gesamte Test etwas mehr als 35 Millionen Jahre dauern.

Ein interessanter Ansatz wird durch das Programm ACOVEA geboten. Dieses benutzt einen Genetischen Algorithmus [GA], dessen Grundidee der biologischen Evolution ähnelt. Dabei wird eine Menge (hier Compileroptionen) von Kombinationsmöglichkeiten (Satz von Compileroptionen) zufällig generiert. Anschließend werden diese Kombinationen nach einem bestimmten Gütekriterium (geringste Laufzeit) selektiert. Daraus (Compileroptionen) werden dann nach dem Zufallsprinzip die Eigenschaften (Hinzufügen oder Wegnehmen von Compileroptionen) verändert und mit anderen kombiniert bzw. gemischt. Auf diese Art und Weise wird eine neue Generation geschaffen, die analog zur vorangegangen behandelt werden kann (Auslese und Kombination). Wird dieser Vorgang einige Male wiederholt, so konvergiert die Lösung in einem lokalen Minimum (geringe Laufzeit). Ein entscheidender Nachteil dieses Verfahrens ist, dass nicht garantiert werden kann, ob das gefundene Minimum das globale oder nur ein lokales Minimum darstellt. Ferner sei das Buch von [Sauer11] (ab Seite 268) und [Ladd95] empfohlen. Im Weiteren soll ACOVEA verwendet werden, um ein Beispielprogramm zu optimieren.

Beispiel 5.8 für „Programmoptimierung mit ACOVEA“
Um die Leistungsfähigkeit von ACOVEA beurteilen zu können, wurde hier anhand einer Fallstudie versucht, einen für die verwendete Testplattform (TP1) geeigneten Satz an Compiler-

optionen zu finden. Hierfür wurde ein Testprogramm erstellt, welches den ungedämpften Duffing Oszillator mit Hilfe des Potenzreihenintegrators löst und dabei den Dezimal Datentyp verwendet. Es wurde die Aufgabenstellung derart gewählt, dass sich eine Ausführungszeit von maximal 5 Millisekunden ergibt. Dafür benötigte ein ACOVEA-Testlauf im Mittel vier Minuten Rechenzeit. Insgesamt wurden vier verschiedene Testläufe durchgeführt, worin ACOVEA jeweils einen Satz an Compileroptionen ermittelte, der zu besserem Laufzeitverhalten führte (verglichen mit den gcc-4.3 Standardoptionen $-Ox$). In Abb. 5.12 auf Seite 128 sind die Laufzeiten (in [ms]) aus dem Protokoll[59] der jeweiligen ACOVEA Testläufe einander gegenübergestellt. Dabei sind für jeden Testlauf die gemessenen Laufzeiten für die Standardoptionen und ACOVEA-Optionen angegeben. Dementsprechend wird ersichtlich, dass ACOVEA in allen vier Fällen eine Kombination von Optionen finden konnte, die zu kürzeren Laufzeiten führte.
In Tabelle 5.5 sind die Compileroptionen für die vier Testläufe eingetragen. Dabei fällt auf, dass ACOVEA bei allen vier Testläufen neun identische Optionen gefunden hat. Außerdem wurden 24 Optionen gefunden, die bei drei Testläufen übereinstimmten. Es scheint also eine Kombination aus Optionen zu geben, mit welcher das Programm noch besser optimiert werden kann. Dies wird hier jedoch nicht weiter verfolgt, da die Leistungsfähigkeit von ACOVEA bereits demonstriert wurde.



	Test 1	Test 2	Test 3	Test 4
Acovea	1,72	1,97	1,70	1,52
O1	3,31	3,32	4,46	2,02
O2	2,55	3,94	3,85	3,90
O3	3,17	3,15	3,17	3,15
Os	2,71	5,15	5,16	2,02

Abbildung 5.12: Resultate der Simulationen mit ACOVEA

Option	T1	T2	T3	T4	Option	T1	T2	T3	T4
-fno-merge-constants	✓			✓	-fno-defer-pop		✓		✓
-fno-inline			✓		-fpeel-loops			✓	✓
-fno-omit-frame-pointer				✓	-fno-tree-ch				✓
-fvariable-expansion-in-unroller			✓		-fbranch-target-load-optimize2				✓
-fbtr-bb-exclusive			✓		-fstrict-aliasing		✓	✓	✓
-freorder-functions			✓	✓	-momit-leaf-frame-pointer			✓	
-minline-all-stringops			✓		-mfpmath=sse,387			✓	
-mfpmath=sse				✓	-fno-thread-jumps	✓	✓		✓

[59]Das Protokoll des Testlaufs Nr. 1 ist exemplarisch in Anhang F.1 auf Seite 215 dargestellt.

-fno-delayed-branch	✓	✓		✓	-fno-loop-optimize		✓		✓
-fstrength-reduce		✓			-frerun-cse-after-loop		✓	✓	✓
-fforce-addr		✓		✓	-fno-tree-dse	✓	✓		
-fno-tree-dominator-opts		✓		✓	-fregmove		✓		✓
-fgcse-las		✓			-fivopts		✓	✓	✓
-fno-tree-ter	✓				-fsched-interblock			✓	✓
-fcaller-saves			✓	✓	-fmodulo-sched		✓	✓	✓
-ftree-vrp		✓	✓	✓	-fno-tree-lrs	✓	✓	✓	✓
-fno-tree-sra		✓			-fno-tree-copyrename		✓		✓
-fno-tree-fre		✓	✓		-fno-move-loop-invariants		✓	✓	✓
-fcrossjumping		✓			-foptimize-sibling-calls	✓	✓	✓	✓
-fcse-follow-jumps		✓			-fgcse		✓	✓	✓
-frerun-loop-opt	✓				-fpeephole2	✓	✓	✓	✓
-fschedule-insns	✓			✓	-fschedule-insns2	✓	✓	✓	
-fsched-spec	✓	✓	✓		-fdelete-null-pointer-checks	✓	✓	✓	
-freorder-blocks	✓	✓	✓		-funit-at-a-time	✓	✓		✓
-falign-jumps	✓		✓		-falign-loops	✓	✓	✓	✓
-falign-labels	✓	✓	✓		-ftree-pre	✓	✓		
-fstrict-overflow	✓	✓	✓	✓	-finline-functions	✓		✓	✓
-funswitch-loops	✓	✓	✓	✓	-fgcse-after-reload	✓	✓		✓
-fpredictive-commoning	✓	✓	✓		-falign-functions	✓	✓	✓	✓
-falign-jumps	✓				-falign-labels	✓			
-fprefetch-loop-arrays	✓	✓		✓	-ftree-vect-loop-version	✓	✓		✓
-ffloat-store	✓	✓			-fprefetch-loop-arrays	✓			
-funswitch-loops	✓				-funroll-loops	✓			✓
-fbranch-target-load-optimize	✓	✓	✓		-fno-function-cse	✓			
-fgcse-sm	✓	✓	✓	✓	-freschedule-modulo-scheduled-loops	✓		✓	✓
-ftree-loop-linear	✓		✓		-ftree-loop-im	✓			
-ftree-loop-ivcanon	✓			✓	-ftree-vectorize	✓	✓	✓	✓
-mieee-fp	✓	✓	✓		-mno-push-args	✓	✓		✓
-maccumulate-outgoing-args	✓				-mno-align-stringops	✓			✓
-finline-limit=800	✓				-msse4.2		✓		

Tabelle 5.5: Ermittelte gcc-4.3 Optimierungsoption von ACOVEA

Weitere Untersuchungen zur automatischen Bestimmung von Compileroptionen wurden von [Granston01] unternommen. Hierbei wurde Hewlett-Packard's PA-RISC Compiler[60] verwendet, um für dessen Optimierungsoptionen ein Optimum anhand der jeweils untersuchten Aufgabenstellung zu finden. Dabei wurde ein Werkzeug zur Optimierung entworfen, welches seine Informationen aus drei verschiedenen Quellen bezieht und basierend auf diesen eine Auswahl geeigneter Optimierungsoptionen trifft. Es werden benutzerdefinierte Optionen, Informationen aus der Übersetzung des Programms und *profiling* Information genutzt, um daraus eine günstigere Kombination an Optimierungsoptionen zu erhalten. Mit dieser kann das Programm erneut übersetzt werden und somit besser auf die jeweilige Zielplattform abgebildet werden, was sich in schnelleren Programmlaufzeiten widerspiegelt. Hierbei konnten bestimmte Unterprogramme des SPEC95-Benchmarks footnotesiehe mehr dazu unter [SPEC95] um das Dreifache schneller ausgeführt werden.
Einen ähnlichen Ansatz hierfür verfolgt [Haneda][61]. Dieser verwendet die Version 3.3.1 der GNU Compiler Collection. Hierbei wurde iterativ vorgegangen und mit einer zufällig generierten Kombination von Compilerflags begonnen. Daraufhin wurde ein Testprogramm übersetzt, dessen Ausführungszeit gemessen und anschließend beurteilt. Dies wurde in einem Beispiel für 5000 verschiedene Kombinationen durch-

[60] siehe mehr dazu unter http://h21007.www2.hp.com, Download am 22.06.2011
[61] siehe mehr dazu unter [Haneda05-1] und [Haneda05-2]

geführt und es konnte gezeigt werden, dass die zufällig erzeugten Kombinationen in vielen Fällen kürzere Laufzeiten erzielen.

5.4.2.1 Fazit

Das Aufspüren der jeweils besten Kombination an Compileroptionen ist ein sehr aufwendiges Unterfangen und verlangt viel Erfahrung im Umgang mit dem verwendeten Compiler. Da sich die verschiedenen Konfigurationsmöglichkeiten bei jeder Version ändern, erfordert dies zusätzliche Einarbeitungszeit. Außerdem optimiert der Compiler das jeweilige Programm auf eine bestimmte Zielplattform, die unter Umständen bestimmte Eigenheiten besitzt. Auch diese sollten die Entwickler kennen um ein Optimum zu erreichen. Falls eine Kombination an Einstellungen gefunden wird, so ist diese optimal auf die verwendete Software zugeschnitten, d.h. wird an der Software etwas verändert, dann muss möglicherweise erneut gesucht werden. Viele der gezeigten Techniken erleichtern es, die Laufzeit eines bestimmten Programms zu vermindern. Dennoch muss bei jedem Anwendungsfall kritisch hinterfragt werden, ob der zeitliche Aufwand, bessere Einstellung zu finden, im jeweiligen Fall gerechtfertigt ist. Außerdem gibt es eine ganze Reihe von Optimierungsoptionen, die durch ungenaueres Rechnen zwar Laufzeit einsparen[62], aber das Endergebnis verfälschen. Diese sollten folglich ignoriert werden. Ferner ist bei der Compileroption *-ansi*[63] Vorsicht geboten. Diese Option sollte während der Entwicklung eines Programms aktiv sein um ANSI/ISO[64] Kompatibilität zu gewährleisten. Beim fertiggestellten Programm sollte sie aber nicht aktiviert werden, da sonst die Laufzeit erheblich verlängert werden kann.
Außerdem soll hier noch auf das Milepost GCC[65]-Projekt hingewiesen werden. Hierbei wurde ein lernender Open-Source-Compiler entwickelt, welcher in der Lage ist, selbstständig zu erkennen, wie Quellcode in Maschinensprache umgesetzt werden muss, um die Zielplattform optimal zu nutzen. Dabei erlernt dieser auf dem GCC basierende Compiler, wie sich auf einer Hardwareplattform die höchste Leistung erzielen lässt. Dazu wurde der Compiler um Schnittstellen ergänzt, über die dieser mit Hilfe von künstlicher Intelligenz Optimierungsvorschläge ermitteln kann. Es ist wünschenswert, dass dieses ambitionierte Projekt in den Hauptentwicklungszweig des GCC-Projekts aufgenommen wird. Somit wäre es für eine breite Nutzergemeinde einsetzbar, was ohne den Aufwand zusätzlicher Softwarepakete, wie ACOVEA, längerfristig zu schnelleren Programmen führen würde.

5.4.3 Weitere geeignete Compiler

In diesem Abschnitt sollen die verfügbaren Compiler für das Linux Betriebssystem bezüglich ihrer Eigenschaften verglichen werden. Hierfür wird zwischen den Sprachen C und C++ und der jeweiligen Eignung des Compilers hinsichtlich der Prozessorarchitektur (32 oder 64 Bit) unterschieden. Diese sind in Tabelle 5.6 aufgelistet. Aus Gründen der Vollständigkeit wurde die GNU Compiler Collection ebenfalls mit in die Liste aufgenommen. Es wird ersichtlich, dass die meisten der ausgewählten Compiler

[62] Beispielsweise **-ffast-math**
[63] siehe mehr dazu unter [Gough04] auf Seite 31
[64] Eine Übersicht zu diesen Standards findet man auf http://www.ansi.org/standards_activities
[65] siehe mehr dazu unter [cTuning],[Fursin10] und [Fursin11]

auch auf 64 Bit Hardwareplattformen und Betriebssystemen einsetzbar sind. Ferner ist noch zu erwähnen, dass die meisten Compiler sowohl die Programmiersprache C als auch C++ unterstützen. Aus diesem Grund sind fast alle, mit Ausnahme von TCC und PCC, für diese Arbeit geeignet. Dabei sei angemerkt, dass die 32 und 64 Bit Kompatibilität des Compilers im Rahmen dieser Arbeit eine untergeordnete Rolle spielt. Zur Entwicklung wurde fast ausschließlich die GCC verwendet. Darum wäre es erstrebenswert herauszufinden, welcher der hier aufgelisteten Compiler das schnellste Programm erzeugt, d.h. es am besten optimiert. Diese Untersuchung wird hier nicht durchgeführt, da dies den Rahmen dieser Arbeit sprengen würde. Eine Untersuchung, bei der drei verschiedene Compiler[66] bezüglich ihres Optimierungsverhaltens verglichen wurden haben gezeigt, dass beim GCC-Compiler viele Optimierungsoptionen keinen Vorteil brachten und es erhebliche Unterschiede zwischen den Compilersuiten gab.

[GCC] GNU Compiler Collection

Name	C	C++	32 Bit	64 Bit	geeignet
GCC	✓	✓	✓	✓	✓
CLang Compiler[67]	✓	✓	✓	✓	✓
Solaris Studio Express [68]	✓	✓	✓	✓	✓
x86Open64Compiler [69]	✓	✓	✓	✓	✓
Open64 Compiler[70]	✓	✓	-	✓	✓
TCC[71]	✓	-	✓	✓	-
PCC[72]	✓	-	✓	✓	-
Intel®Compiler [73]	✓	✓	✓	✓	✓
Rose Compiler[74]	✓	✓	✓	✓	✓
PathScale EKO Compiler[75]	✓	✓	-	✓	✓
Open Watcom Compiler [76]	✓	✓	✓	-	✓

Tabelle 5.6: Gegenüberstellung verschiedener Compiler für Linux mit deren Eigenschaften

In Tabelle 5.6 sind ausschließlich *Open Source*-Projekte aufgeführt, an denen aktiv entwickelt wird[77]. Es gibt eine Vielzahl weiterer, prinzipiell erwähnenswerter *Open Source*-Compiler Projekte, deren Weiterentwicklung meist aus Zeitmangel der Beteiligten eingestellt wurde. Da in dieser Arbeit das Augenmerk auf optimierten Programmen liegt, sind Compiler dieser Art i.d.R. nicht relevant und werden hier bewusst nicht weiter erwähnt.

[66] Siehe mehr dazu unter http://www.heise.de/developer/meldung/Compilervergleich-Intel-vor-Microsoft-vor-GCC-1469149.html, Download am 10.04.2012
[67] siehe mehr dazu unter [CLangCompiler]
[68] siehe mehr dazu unter [SolarisStudioExpress]
[69] siehe mehr dazu unter [x86Open64Compiler]
[70] siehe mehr dazu unter [Open64]
[71] siehe mehr dazu unter [TCC]
[72] siehe mehr dazu unter [PCC]
[73] siehe mehr dazu unter [IntelCompiler]
[74] siehe mehr dazu unter [RoseCompiler]
[75] siehe mehr dazu unter [PathScaleEKOCompiler]
[76] siehe mehr dazu unter [OpenWatcom]
[77] Stand Juli 2011

Kapitel 6

Methoden und Aspekte numerischer Differentiation

Schwerpunkt des Kapitels:

In diesem Abschnitt werden Aspekte der numerischen Differentiation betrachtet und insbesondere Formeln zur Berechnung höherer Ableitungen entwickelt. Diese werden beispielsweise benötigt, falls keine analytische Darstellung einer Funktion vorliegt oder die höheren Ableitungen einer gesuchten Funktion zu aufwendig zu berechnen sind. Darüber hinaus können die nachfolgend erarbeiteten Formeln zur Verifikation analytisch berechneter höherer Ableitungen herangezogen werden. Ferner wird die Technik des symbolischen Differenzierens und die Methode des automatischen Differenzierens anhand des Keplerproblems beispielhaft untersucht und bewertet.

6.1 Bestimmung von höheren Ableitungen mit Hilfe des Differenzenquotienten

Soll eine Funktion $f(x)$ numerisch differenziert werden, so kann der so genannte Differenzenquotient zur Berechnung der ersten Ableitung einer stetig differenzierbaren Funktion verwendet werden. Dabei ist jedoch Vorsicht geboten, da hierbei lediglich eine angenäherte Lösung berechnet wird. Die verschiedenen Varianten von Differenzenquotienten können formell wie folgt definiert werden:

- Der rechtsseitige Differenzenquotient:

$$f'(x_0) \approx \lim_{x \to 0} \frac{f(x_0 + h) - f(x_0)}{h} \tag{6.1.1}$$

- Der linksseitige Differenzenquotient:

$$f'(x_0) \approx \lim_{x \to 0} \frac{f(x_0) - f(x_0 - h)}{h} \tag{6.1.2}$$

- Der zentrale Differenzenquotient:

$$f'(x_0) \approx \lim_{x \to 0} \frac{f(x_0 + h) - f(x_0 - h)}{2h} \tag{6.1.3}$$

In Abb. 6.1 sind die unterschiedlichen Varianten der Differenzenquotienten geometrisch interpretiert. Dabei ist h der Abstand der zur Berechnung verwendeten Funktionswerte. Daraus wird deutlich, dass zwischen zwei Punkten ein linearer Zusammenhang unterstellt wird, der unter Umständen nicht mit dem tatsächlichen Funktionswert zusammenstimmt.

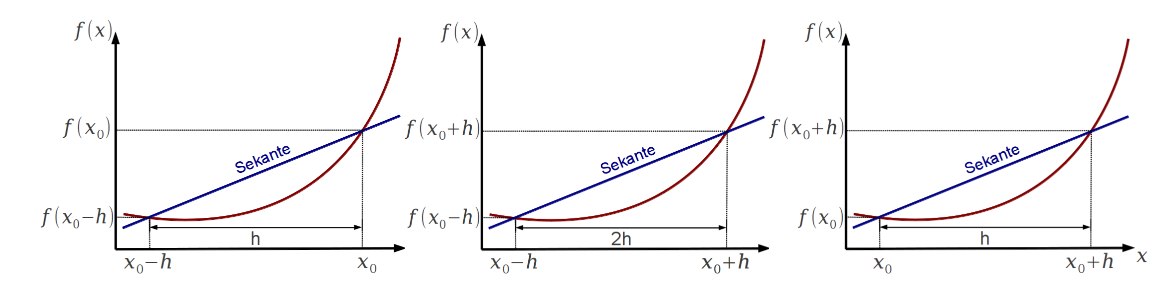


Abbildung 6.1: Linksseitiger, zentraler und rechtsseitiger Differenzenquotient

6.2 Lösungsformeln zur numerischen Berechnung höherer Ableitungsterme

In diesem Abschnitt sollen auf numerischem Weg höhere Ableitungen berechnet werden. Diese werden insbesondere benötigt, falls für eine gesuchte Funktion keine analytische Lösung vorhanden ist oder diese zu aufwendig zu berechnen ist. Dadurch

scheitert der intuitive Ansatz durch Berechnung des Differenzenquotienten häufig, besonders wenn die verwendeten Stützstellen zu nahe beisammen liegen und dadurch Auslöschungseffekte[1] auftreten.

Begriffsbestimmung BEG6.1 „Numerische Auslöschung“:
Dies ist ein Effekt, der bei Subtraktion ähnlicher Gleitkommazahlen auftreten kann. Bei komplexen Berechnungen kann nicht vorhergesagt werden, ob dieser Effekt auftritt und mit welchem Genauigkeitsverlust zu rechnen ist. Konkret tritt dieser Effekt auf, falls zwei Gleitkommazahlen fast identische Mantissenbits besitzen und diese subtrahiert werden. Dadurch werden die identischen Bits ausgelöst, was i.d.R. ungenaue Ergebnisse verursacht. Falls sich beispielsweise nur die letzten Bits unterscheiden, so enthält nach der Subtraktion der beiden Zahlen die Mantisse des Ergebnisses im ungünstigsten Fall nur noch ein gültiges Bit. Ob dieser Effekt auftritt, hängt folglich von den jeweils subtrahierten Zahlen und der Anzahl der Mantissenbits ab.

Beispiel 6.1 „Entwicklung einer Lösungsformel zur numerischen Berechnung der ersten und zweiten Ableitung“
Hier wird exemplarisch eine Lösungsformel zur Berechnung der ersten und zweiten Ableitung einer gegebenen Funktion $f(x)$ erarbeitet. Im ersten Schritt wird von einer Taylorreihe 4.Grades ausgegangen:

$$f(x) = f(x_0) + f^{'}(x_0) + \frac{1}{2}f^{''}(x_0)h^2 + \frac{1}{6}f^{(3)}(x_0)h^3 + \frac{1}{24}f^{(4)}(x_0)h^4 + O(h^5) \quad (6.2.4)$$

Es werden zusätzlich zwei Funktionswerte benötigt. Es werden die Werte $f(x_0+h)$ und $f(x_0-h)$ gewählt, mit einem Abstand von $2h$. Werden die beiden Funktionswerte eingesetzt, dann können die beiden Taylorreihen wie folgt aufgestellt werden:

$$\begin{aligned} f(x_0-h) &= f(x_0) - f^{'}(x_0) + \frac{1}{2}f^{''}(x_0)h^2 - \frac{1}{6}f^{(3)}(x_0)h^3 + \frac{1}{24}f^{(4)}(x_0)h^4 \\ f(x_0+h) &= f(x_0) + f^{'}(x_0) + \frac{1}{2}f^{''}(x_0)h^2 + \frac{1}{6}f^{(3)}(x_0)h^3 + \frac{1}{24}f^{(4)}(x_0)h^4 \end{aligned} \quad (6.2.5)$$

Werden die beiden Reihen als ein Gleichungssystem mit den Unbekannten $f^{'}(x_0)$ und $f^{''}(x_0)$ angesehen, so kann es formal nach den Unbekannten aufgelöst werden. Die höheren Ableitungen $f^{(3)}$ und $f^{(4)}$ sind in diesem Zusammenhang als Konstanten zu betrachten.

$$\begin{aligned} f'(x_0) &= \frac{f(x_0+h) - f(x_0-h)}{2h} + \frac{1}{6}h^2 f^{(3)}(x_0) + O(h^2) \\ f^{''}(x_0) &= \frac{f(x_0+h) - 2f(x_0) + f(x-h)}{h^2} - \frac{1}{12}h^2 f^{(4)}(x_0) + O(h^4) \end{aligned} \quad (6.2.6)$$

Der erste Term der Gleichung 6.2.6 entspricht der Lösungsformel zur Berechnung der ersten Ableitung (siehe 6.1). Mit dieser Vorgehensweise können Lösungsformeln für die Ableitungen $f^{(1)}, ..., f^{(n)}; \forall n \epsilon N \geq 1$, gebildet werden.

Wie gezeigt wurde, können prinzipiell für beliebig hohe Ableitungen Lösungsformeln erarbeitet werden. Um diese Formeln zu berechnen, werden mehrere Funktionswerte benötigt und es muss ein entsprechend höherer Entwicklungsgrad bei der Taylorreihenentwicklung gewählt werden. Im nächsten Unterpunkt werden in analoger Vorgehensweise Lösungsformeln erarbeitet, mit denen Ableitungen bis zur 18.Ablei-

[1] Ein Beispiel zur damit verbundenen Fehlerrechnung ist in [Press07] auf Seite 229 (unten) zu finden.

tung berechnet werden können.

Beispiel 6.2 für „Erstellung von Lösungsformeln zur numerischen Berechnung höherer Ableitungen“
Das Erarbeiten von Lösungsformeln für höhere Ableitungen ohne technische Hilfsmittel ist zwar prinzipiell möglich, würde aber zu lange dauern. Aus diesem Grund wird auf ein Hilfsprogramm zurückgegriffen, das von [Mathews03] entwickelt wurde. Dieses Programm wurde mit MATHEMATICA[2] *entwickelt und verwendet den zentralen Differenzenquotienten zur Ermittlung der höheren Ableitung. Die Vorgehensweise zur Bestimmung der Lösungsformeln ist dabei analog zur bereits gezeigten (siehe Abschnitt 6.2).*

```
CentralDiff[k0_] := Module[{k = k0},
 Clear[h, i, j, n, f, s, x];
 s[h_] = Normal[Series[f[x0 + h], {h, 0, 2 k + 2}]];
 eqns = Table[s[j h] == f[x0 + j h], {j, -k, k}];
 vars = Table[f^(k)[x0], {k, 1, 2 k}];
 pts = Table[{x0, f[x0 + j h]}, {j, -k, k}];
 solset = Solve[eqns, vars];
 solset[[1,All,2]] = Collect[ExpandAll[solset[[1,All,2]]], h];
 solset[[1,All,2,2]] = ReplaceAll[solset[[1,All,2,2]], f^(i_)[x0] → f^(i)[c]];
 solset[[1,All,2,1]] = Together[solset[[1,All,2,1]]];
 Print[
  "Central difference formulas for numerical differentiation"];
 Print["Using the ", 2 k + 1, " points"];
 Print[ pts, "\n" ];
 Print[TableForm[solset[[1]]]]; ];
```

Abbildung 6.2: Mathematica Funktion zur symbolischen Berechnung von Formeln zur Bestimmung höherer Ableitungen

Die Funktion, welche zur Berechnung herangezogen wurde, ist aus Gründen der Vollständigkeit in Abb.6.2 dargestellt. Dieses Programm benötigt zur Berechnung lediglich die Anzahl $\frac{m-1}{2}$ *der zu verwendeten Stützstellen. Es wird jeweils eine ungerade Anzahl von Stützstellen verwendet, da der symmetrische Differenzenquotient und der Entwicklungspunkt der Taylorreihe benötigt wird. Falls beispielsweise fünf Stützstellen* ($m = 5$) *verwendet werden sollen, ist die Funktion mit* CentralDiff[2] *aufzurufen* ($k = 2$)*. Somit ergibt sich folgender Zusammenhang:*

$$m = 2k + 1 \tag{6.2.7}$$

Um eine Lösungsformel für die 18.Ableitung zu erhalten, muss das Programm konsequenterweise mit CentralDiff[9] *aufgerufen werden. Damit werden folgende 19 Funktionswerte zur Berechnung der höheren Ableitungen benötigt:*

$$\begin{gathered} f(-9h + x_0), f(-8h + x_0), f(-7h + x_0), f(-6h + x_0) \\ f(-5h + x_0), f(-4h + x_0), f(-3h + x_0), f(-2h + x_0) \\ f(h + x_0), f(+x_0), f(h - x_0) \\ f(2h + x_0), f(3h + x_0), f(4h + x_0), f(5h + x_0) \\ f(6h + x_0), f(7h + x_0), f(8h + x_0), f(9h + x_0) \end{gathered}$$

[2] siehe mehr dazu unter http://www.wolfram.com/mathematica/

Im Folgenden werden die Formeln für die ersten vier Ableitungen, basierend auf 19 Funktionswerten dargestellt. Alle weiteren Formeln sind im Anhang E auf Seite 209 zu finden.

$$
\begin{aligned}
f'(x_0) = \frac{1}{12252240h}(&-28f(-9h+x_0) + 567f(-8h+x_0) - 5508f(-7h+x_0)\\
&+ 34272f(-6h+x_0) - 154224f(-5h+x_0) + 539784f(-4h+x_0)\\
&- 1559376f(-3h+x_0) + 4009824f(-2h+x_0) - 11027016f(-h+x_0)\\
&+ 11027016f(h+x_0) - 4009824f(2h+x_0) + 1559376f(3h+x_0)\\
&- 539784f(4h+x_0) + 154224f(5h+x_0) - 34272f(6h+x_0)\\
&+ 5508f(7h+x_0) - 567f(8h+x_0) + 28f(9h+x_0))
\end{aligned}
$$

$$
\begin{aligned}
f''(x_0) = \frac{1}{15437822400h^2}(&-47541321542f(x_0) + 7840f(-9h+x_0)\\
&- 178605f(-8h+x_0) + 1982880f(-7h+x_0) - 14394240f(-6h+x_0)\\
&+ 77728896f(-5h+x_0) - 340063920f(-4h+x_0) + 1309875840f(-3h+x_0)\\
&- 5052378240f(-2h+x_0) + 27788080320f(-h+x_0) + 27788080320f(h+x_0)\\
&- 5052378240f(2h+x_0) + 1309875840f(3h+x_0) - 340063920f(4h+x_0)\\
&+ 77728896f(5h+x_0) - 14394240f(6h+x_0) + 1982880f(7h+x_0)\\
&- 178605f(8h+x_0) + 7840f(9h+x_0))
\end{aligned}
$$

$$
\begin{aligned}
f^{(3)}(x_0) = \frac{1}{3027024000h^3}(&63397f(-9h+x_0) - 1281033f(-8h+x_0)\\
&+ 12405267f(-7h+x_0) - 76813928f(-6h+x_0) + 342868500f(-5h+x_0)\\
&- 1182036366f(-4h+x_0) + 3302404924f(-3h+x_0) - 7666346376f(-2h+x_0)\\
&+ 8823005334f(-h+x_0) - 8823005334f(h+x_0) + 7666346376f(2h+x_0)\\
&- 3302404924f(3h+x_0) + 1182036366f(4h+x_0) - 342868500f(5h+x_0)\\
&+ 76813928f(6h+x_0) - 12405267f(7h+x_0) + 1281033f(8h+x_0)\\
&- 63397f(9h+x_0))
\end{aligned}
$$

$$
\begin{aligned}
f^{(4)}(x_0) = \frac{1}{54486432000h^4}(&842762184550f(x_0) - 507176f(-9h+x_0)\\
&+ 11529297f(-8h+x_0) - 127597032f(-7h+x_0) + 921767136f(-6h+x_0)\\
&- 4937306400f(-5h+x_0) + 21276654588f(-4h+x_0) - 79257718176f(-3h+x_0)\\
&+ 275988469536f(-2h+x_0) - 635256384048f(-h+x_0) - 635256384048f(h+x_0)\\
&+ 275988469536f(2h+x_0) - 79257718176f(3h+x_0) + 21276654588f(4h+x_0)\\
&- 4937306400f(5h+x_0) + 921767136f(6h+x_0) - 127597032f(7h+x_0)\\
&+ 11529297f(8h+x_0) - 507176f(9h+x_0))
\end{aligned}
$$

Beispiel 6.3 für „Verifikation der Lösungsformeln anhand des Duffing-Oszillators“
In diesem Beispiel werden die erarbeiteten Lösungsformeln anhand der Ableitungen des ungedämpften Duffing-Oszillators validiert. Hierzu werden die dafür benötigten numerischen Werte der Ableitungsterme mit dem Potenzreihenintegrator mit entsprechend hoher Genauigkeit berechnet. Diese können zum Vergleich herangezogen werden und in diesem Zusammenhang als Referenzlösung angesehen werden. Ferner besteht bei den Formeln eine zusätzliche Abhängigkeit zum Parameter h. Dieser beeinflusst die Qualität des Ergebnisses, wie aus Abb. 6.3 ersichtlich wird.

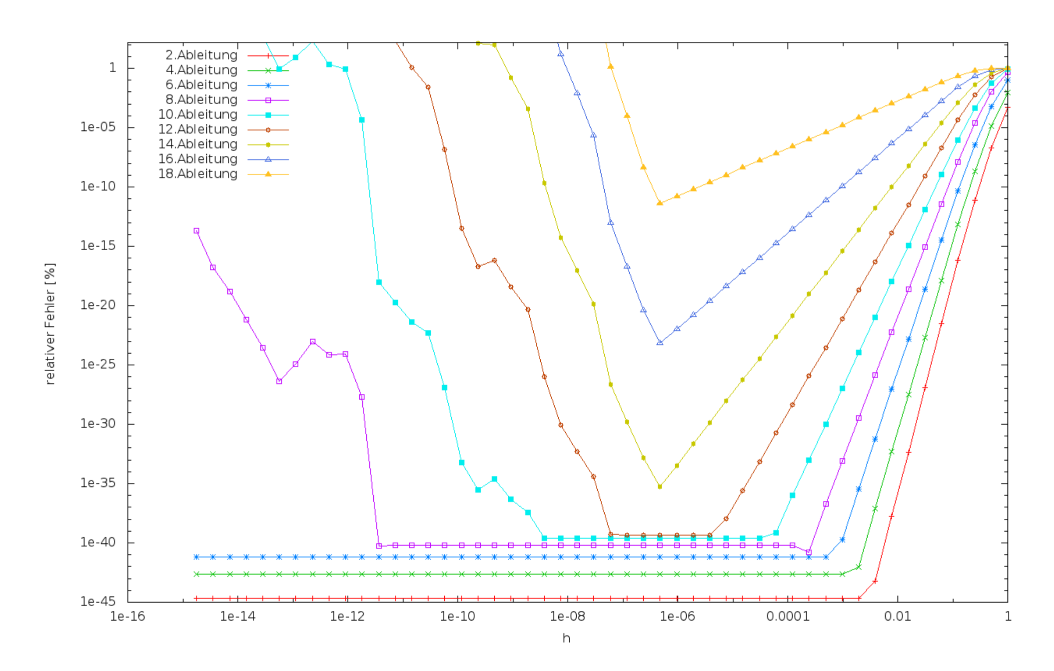


Abbildung 6.3: Überprüfung der Lösungsformeln mit unterschiedlicher Schrittweite

Hier wurden alle geraden Ableitungsterme[3] *bis zur 18.Ableitung der Referenzlösung gegenübergestellt. Um die Abhängigkeit des Parameters* h *sichtbar zu machen, wurde dieser sukzessive verkleinert (siehe x-Achse). Aus dem relativen Fehler (y-Achse) der jeweiligen Berechnung wird ersichtlich, dass es für jeden Ableitungsterm ein optimales* h *gibt, welches die besten Ergebnisse liefert. Die Herausforderung besteht nun darin, ein optimales* h *zu finden, um den relativen Fehler pro Ableitungsterm zu minimieren. Außerdem ist auffällig, dass die erreichbare Genauigkeit mit steigendem Grad der Ableitung abnimmt. Um dies zu überprüfen, wurde ein weiterer Satz an Formeln erstellt, welcher 13 Punkte zur Berechnung eines Ableitungswertes benötigt. Die Ergebnisse aus dem Vergleich mit der Referenzlösung sind in Abbildung 6.4 dargestellt. Darin ist ebenfalls eine Abnahme der relativen Genauigkeit bei zunehmenden Ableitungsgrad zu erkennen.*

[3] Durch die gewählten Startwerte ($u = 1, \dot{u} = 0$) des ungedämpften Duffing-Oszillators haben alle ungeraden Ableitungsterme den Wert Null und werden nicht berücksichtigt.

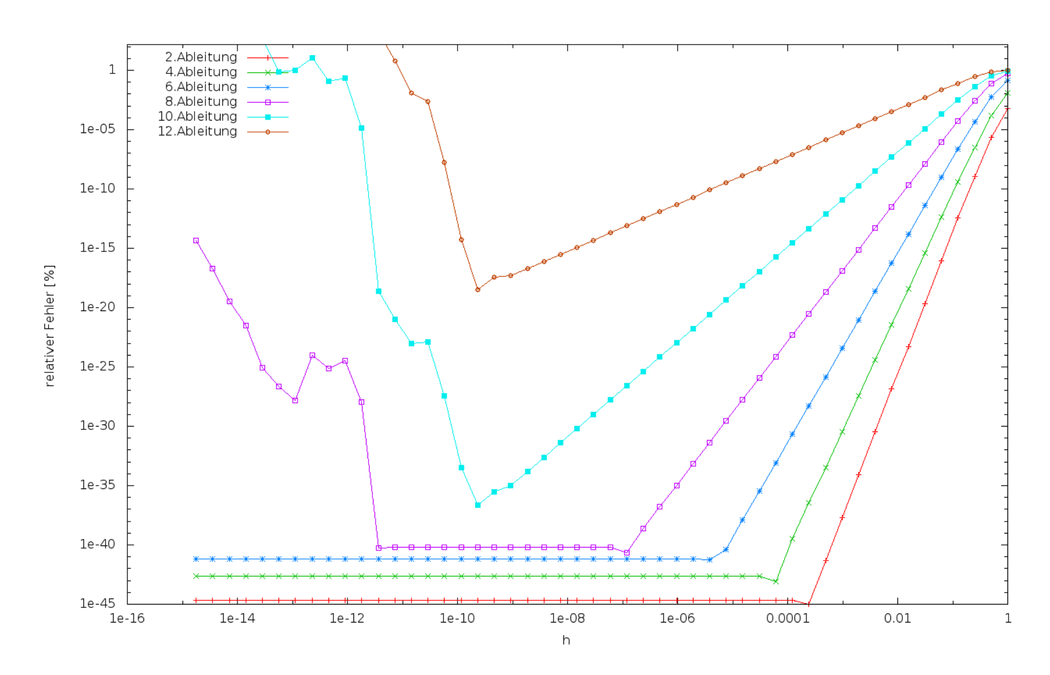


Abbildung 6.4: Überprüfung der Lösungsformel mit unterschiedlicher Schrittweite

Beispiel 6.4 für „Algorithmus zur Verbesserung der Lösung“
Es soll ein Algorithmus angegeben werden, mit dem die Ableitungswerte noch verbessert werden können. Wie im vorangegangenen Beispiel gezeigt wurde, hängt die Qualität des Ergebnisses einer bestimmten Ableitung von der gewählten Schrittweite h ab. Dieses Problem wurde in ähnlicher Form schon von [Ridders82] gelöst, jedoch nur für die Ableitungen ersten und zweiten Grades. Aus dieser Vorgehensweise soll nun eine Methode für höhere Ableitungen erstellt werden. Hierzu wird analog zum Vorgehen von [Ridders82] in einer Anlaufphase ein Satz von Ableitungswerten $f^{(i)}(i \in N^+)$ berechnet, wobei bei jeder der Berechnungen die Schrittweite h halbiert wird. Nach dieser Anlaufphase liegt folgender Satz an Ableitungswerten vor:

$$\underbrace{f^{(i)}(\frac{h}{1})}_{A_1}, \underbrace{f^{(i)}(\frac{h}{2})}_{A_2}, \underbrace{f^{(i)}(\frac{h}{4})}_{A_3}, \underbrace{f^{(i)}(\frac{h}{8})}_{A_4}, ..., \underbrace{f^{(i)}((\frac{h}{2})^N)}_{A_N} \tag{6.2.8}$$

wobei N ein Platzhalter für die Anzahl der Unterteilungen während der Anlaufphase ist. Abkürzend kann der Satz an Ableitungswerten als $A_1, ..., A_N$ bezeichnet werden[4]. Im nächsten Schritt wird auf die Methode der Richardson Extrapolation zurückgegriffen, welche auch beim Integrationsverfahren von Burlisch und Stoer verwendet wird. Dieses Verfahren, einschließlich der Richardson Extrapolation, ist in Abschnitt 2.3.5.3 auf Seite 25 beschrieben. Als nächstes können zwei aufeinander folgende Werte verwendet werden, um daraus mit Hilfe der Richardson Extrapolationsmethode eine Verbesserung B zu erzielen. Das kann beispielsweise für die erste Verbesserung wie folgt durchgeführt werden[5]:

$$B_1 = \frac{A_2 h^m - A_1}{h^m - 1}, m = 1 \tag{6.2.9}$$

Werden die Verbesserungen sukzessive durchgeführt, dann ergibt sich das in Abbildung 6.5

[4]Hierbei wird die Nomenklatur von [Ridders82] übernommen.
[5]Es wird hier lediglich die Formel der Richardson-Extrapolation auf diese Problemstellung angewendet. Siehe mehr dazu in [Ridders82].

schematisch dargestellte Schema[6]*. Hierbei ist der Buchstabe C ein Platzhalter für einen weiteren Verbesserungszyklus.*

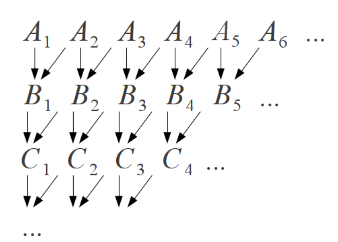


Abbildung 6.5: Schema zur Verbesserung der Lösung durch Richardson Extrapolation

Um ein Abbruchkriterium zu erhalten, sind im Fall der ersten Verbesserungsiteration (B) folgende zwei Fehlerbeträge zu bestimmen:

$$\begin{aligned} \epsilon_1 &= |B_2 - B_1| \\ \epsilon_2 &= |B_2 - A_1| \end{aligned} \tag{6.2.10}$$

Anschließend wird der größere der beiden Werte mit dem ϵ der letzten Iteration verglichen. Ist dieses ϵ kleiner, so wurde die Lösung verbessert und kann zwischengespeichert werden. Im anderen Fall wurde keine Verbesserung erzielt, die Lösung wird verworfen und eine neue Iteration begonnen. Als Anfangsbedingung wird hierbei, analog zur Implementierung in [Press07][7]*, der Wert von $\epsilon = 1.797693e + 308$ gesetzt. In Abb. 6.6 sind die Ergebnisse der Verbesserungen der beiden Formeln, bestehend aus 13 und 19 Stützstellen, dargestellt. Bei den jeweils verbesserten Lösungen fällt auf, dass die Richardson Extrapolationsmethode gerade bei hohen Ableitungen die Ergebnisse derart gut verbessert, dass der relative Fehler stets im Bereich von 1.0E-40 bleibt. Bei den nicht verbesserten Lösungen wurde dieselbe Schrittweite verwendet, welche durch das Verbesserungsverfahren ermittelt wurde, jedoch verschlechterte sich das Ergebnis mit zunehmenden Grad der Ableitung. Dieser Trend konnte bereits aus Abb. 6.3 und 6.4 abgeleitet werden.*

[6] Diese ist in ähnlicher Weise in [Scheid88] auf Seite 114 (Abschnitt 13.30) abgedruckt.

[7] Hier wird eine ähnliche Implementierung angegeben. Sie mehr dazu auf Seite 253. Die dort definierte Funktion lautet *dfridr*.

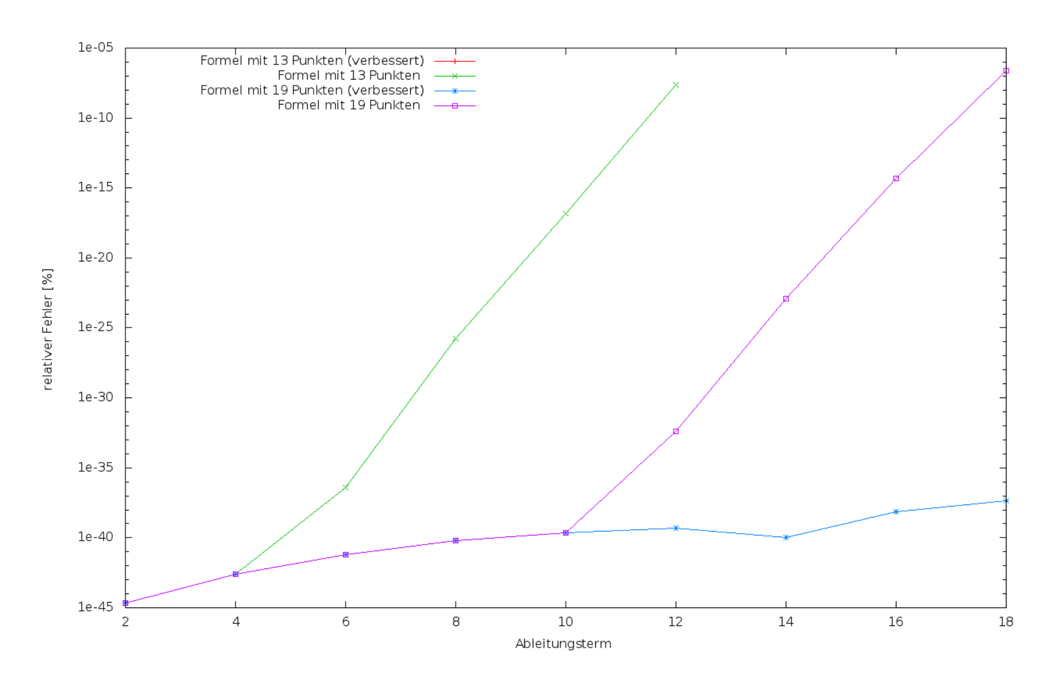


Abbildung 6.6: Verbesserung durch Richardson Extrapolation

6.3 Weitere Methoden der Differentiation

In diesem Abschnitt sollen weitere Methoden der Differentiation betrachtet werden. Neben den bereits erarbeiteten Lösungsformeln kann durch symbolisches Differenzieren - eine gegebene Funktionsdefinition vorausgesetzt - die formelmäßige Darstellung der Ableitungsfunktion[8] streng mathematisch berechnet werden.
Ist die Aufgabenstellung bereits als Computerprogramm realisiert, so bietet sich automatisches Differenzieren (AD) an. Hierbei werden nicht wie beim numerischen Differenzieren die entsprechenden Differenzenquotienten berechnet, sondern es wird aus dem bestehenden Computerprogramm eine erweiterte Funktion generiert. Mit Hilfe dieser generierten Erweiterung ist es prinzipiell möglich, beliebig hohe Ableitungsterme zu errechnen. Diese generierte Funktion besteht ausschließlich aus mathematischen Grundfunktionen (z.B. $sin, cos, ...$), deren Ableitungen bekannt sind und exakt berechnet werden können. Durch Anwenden der Kettenregel auf die Ausgangsfunktion[9] und mit dem Wissen, wie die Ableitungen der Grundfunktionen beschaffen sind, lassen sich somit Funktionen generieren, die das Ergebnis der jeweils gewünschten Ableitung liefern.
Der Zusammenhang zwischen den Techniken der symbolischen und automatischen Differentiation ist in Abb. 6.7 dargestellt. Wird eine symbolisch vorhandene Funktion differenziert, so wird diese in vielen Fällen in Quelltext (Computerprogramm) übertragen. Dieser Vorgang ist bei umfangreichen Funktionen besonders zeitaufwendig und führt dabei oft zu Übertragungsfehlern. Das Auffinden dieser kann unter Umständen viel Zeit in Anspruch nehmen. Im Gegensatz dazu bietet das automatische Differenzie-

[8]Dies setzt eine stetig differenzierbare Funktion der untersuchten Problemstellung voraus.
[9]Zusätzliche Informationen zum automatischen Differenzieren gibt es auf folgender Webseite: http://www.autodiff.org/

ren die Möglichkeit, ohne derartige Übertragungsfehler eine funktionale Implementierung der Ableitungsfunktion zu generieren.

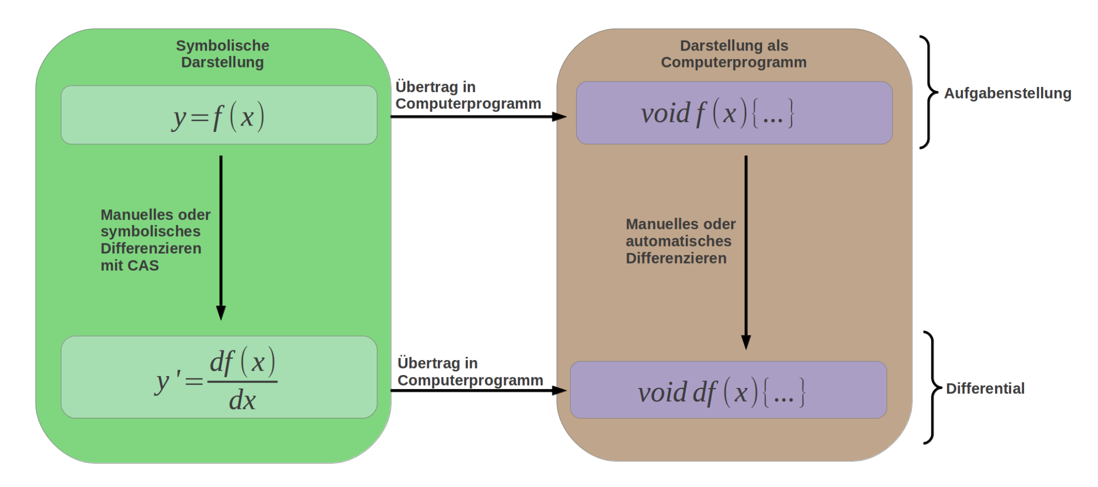


Abbildung 6.7: Zusammenhang zwischen symbolischer Ableitung und Bestimmung einer Ableitung mittels automatischer Differentiation

Beispiel 6.5 für „Berechnung höherer Ableitungen des Keplerproblems mit Hilfe symbolischer Rechnung“
In diesem Beispiel soll die dritte Ableitung des Keplerproblems auf symbolischem Weg berechnet werden. Die folgenden Variablen werden als gegeben vorausgesetzt:

$$\vec{r} = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix}; \quad r = \sqrt{x^2(t) + y^2(t) + z^2(t)};$$

$$\dot{\vec{r}} = \begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{z}(t) \end{pmatrix}; \quad \dot{r} = \frac{dr}{dt} = \frac{2z(t)\left(\frac{d}{dt}z(t)\right) + 2y(t)\left(\frac{d}{dt}y(t)\right) + 2x(t)\left(\frac{d}{dt}x(t)\right)}{2\sqrt{z^2(t) + y^2(t) + x^2(t)}} = \frac{[\vec{r}\dot{\vec{r}}]}{r};$$

$$\mu = GM_{\oplus};$$

dabei ist:

- $\vec{r}$ *der Ortsvektor im Inertialsystem und* r *die geometrische Distanz.*
- $\dot{\vec{r}}$ *der Vektor der Geschwindigkeitskomponenten und* $\dot{r}$ *die Radialgeschwindigkeit.*
- μ *die Gravitationskonstante* G *multipliziert mit der Masse der Erde* $M_{\oplus}$.
- t *die Zeit.*

Durch Anwenden der bekannten Regeln der Differentiation[10] *ergeben sich folgende Ergebnisse*[11]*:*

$$\frac{d^0\vec{r}}{dt^0} = \vec{r}$$
$$\frac{d^1\vec{r}}{dt^1} = \dot{\vec{r}}$$
$$\frac{d^2\vec{r}}{dt^2} = -\mu\frac{\vec{r}}{r^3}$$
$$\frac{d^3\vec{r}}{dt^3} = -\mu[\frac{\dot{\vec{r}}}{r^3} - \frac{3\vec{r}[\vec{r}\dot{\vec{r}}]}{r^5}]$$

Nachdem die Ableitungsterme formelmäßig bestimmt sind, können sie in ein Computerprogramm übertragen werden. Eine mögliche Implementierung der Kraftfunktion und der 3. Ableitung ist in Listing 6.1 und 6.2 angegeben.

```
 1 #include <math.h>
 2 #define GME 0.3986004415000000E+6 /* [km^3/s] */
 3 void Kepler(double *x, double *y, double *z,
 4        double *vx, double *vy, double *vz)
 5 {
 6   double r, _x, _y, _z;
 7   _x = *x; _y = *y; _z = *z;
 8   r =  sqrt( (_x) * (_x) + (_y) * (_y) + (_z) * (_z) );
 9   r *= r*r;
10   *x  = *vx;
11   *y  = *vy;
12   *z  = *vz;
13   *vx = - GME * _x /r;
14   *vy = - GME * _y /r;
15   *vz = - GME * _z /r;
16 }
```

Listing 6.1: Implementierung der Keplerkraftfunktion ($\frac{d^2\vec{r}}{dt^2}$) in der Programmiersprache C aus symbolischer Rechnung

```
 1 void dKepler(double *x, double *y, double *z, double vx, double vy,
      double vz)
 2 {
 3   double r, r3, r5 , rv;
 4   r =  sqrt( (*x) * (*x) + (*y) * (*y) + (*z) * (*z) );
 5   rv = ( (*x * vx) + (*y * vy) + (*z * vz) );
 6   r3 = r*r*r;
 7   r5 = r3*r*r;
 8   *x = - GME * ( (vx)/r3 - 3*(*x) * rv/r5 ) ;
 9   *y = - GME * ( (vy)/r3 - 3*(*y) * rv/r5 ) ;
10   *z = - GME * ( (vz)/r3 - 3*(*z) * rv/r5 ) ;
11 }
```

Listing 6.2: Implementierung der 3. Ableitung ($\frac{d^3\vec{r}}{dt^3}$) in der Programmiersprache C aus symbolischer Rechnung

[10] Produkt -u. Kettenregel der Differentialrechnung: Siehe mehr dazu in [Bronstein08] auf Seite 435 und 436.

[11] Im Anhang C (Seite 191) sind Formeln bis zur 10.Ableitung angegeben.

Eine alternative Möglichkeit, eine Implementierung der 3. Ableitung zu erhalten, stellt automatisches Differenzieren dar.

Beispiel 6.6 für „Bestimmung der 3.Ableitung durch automatisches Differenzieren“

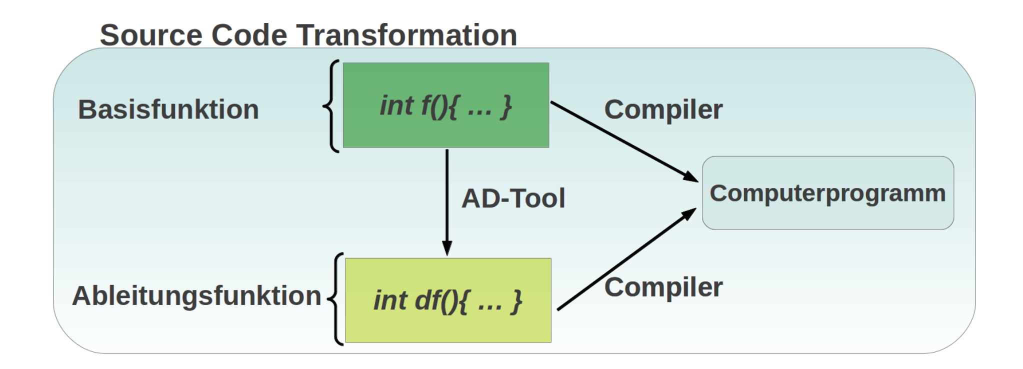
[AD] Automatisches Differenzieren

Als Ausgangspunkt zum AD wird die abzuleitende Funktion in einer i.d.R. frei wählbaren Programmiersprache als Funktion implementiert. Diese wird, im Sinne der Differentiation, gewöhnlich als Basisfunktion bezeichnet. Zur Bestimmung der Ableitungsfunktion aus der Basisfunktion stehen prinzipiell zwei unterschiedliche Algorithmen zur Verfügung, der Vorwärts - u. Rückwärtsmodus[12]*. Beide Algorithmen basieren dabei auf der Grundidee, aus einer Basisfunktion, welche aus mathematischen Grundoperationen aufgebaut ist, eine Ableitungsfunktion zu generieren. Wird der Vorgang der Ableitung als Aufrufgraph angesehen, so unterscheiden sich die beiden Ansätze lediglich in der Abarbeitungsreihenfolge. Darüber hinaus gibt es zwei unterschiedliche Vorgehensweisen, eine automatische Differentiation durchzuführen:*

1. Source Code Transformation *[SCT]*
   Hierbei wird die Basisfunktion lediglich eingelesen und daraus eine Ableitungsfunktion generiert. Diese minimal invasive Technik kann prinzipiell auf eine beliebige Programmiersprache angewendet werden.

2. Operatorüberladung
   Diese Methode erfordert u.U. die Veränderung des Quelltextes, falls dieser nicht objektorientiert programmiert wurde. Außerdem ist eine Änderung einiger Datentypen auf den Dualzahlen-Datentyp erforderlich. Da dieser kein Standarddatentyp der C-Standardbibliothek ist, enthalten die meisten AD-Bibliotheken eigene Implementierungen für Dualzahlen. Die Vorgehensweise zum Differenzieren ist in Abb. 6.8 schematisch dargestellt.

[12] Siehe mehr dazu in [Bücker06]



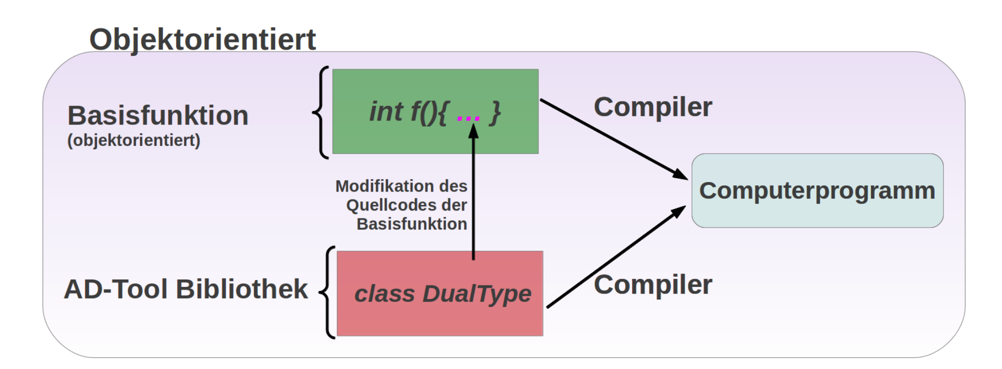


Abbildung 6.8: Schematische Darstellung der Vorgehensweise beim automatischen Differenzieren mit objektorientiertem und SCT-Ansatz

Die Aufgabenstellung des vorausgegangenen Beispiels wird nun wieder aufgegriffen und mittels SCT-Vorwärtsmodus wird eine Ableitungsfunktion bestimmt. Hierfür wurde der Online-Dienst TAPENADE[13] *verwendet. Das Resultat der Quelltexttransformation ist in Listing 6.3 dargestellt*[14]*. Vergleicht man den Quelltext der automatischen Differentiation mit dem der symbolisch bestimmten Lösung, so wird ersichtlich, dass die generierte Lösung mehr Rechenoperationen benötigt. Verglichen mit manueller Implementierung und dem damit verbundenen Zeitaufwand, ist diese Methode erheblich effizienter. Im Anhang G.11 (Seite 231) ist der Quelltext für den Rückwärtsmodus angegeben. Beim Vergleich der resultierenden Ableitungsfunktionen, welche im Vor- bzw. Rückwärtsmodus gebildet wurden, fällt auf, dass die Anzahl der benötigten arithmetischen Operationen für die Berechnung des Vorwärtsmodus geringer ist.*

```
1 void dKepler(double *x, double *xd, double *y, double *yd, double *z,
       double *zd, double *vx, double *vxd, double *vy, double *vyd,
      double *vz, double *vzd)
2 {
3     double r, _x, _y, _z;
4     double rd, _xd, _yd, _zd;
5     double arg1, arg1d , GME;
6     GME = 0.3986004415000000E+6;
7     _xd   = *xd;   _x = *x;
8     _yd   = *yd;   _y = *y;
9     _zd   = *zd;   _z = *z;
```

[13]Siehe mehr dazu unter http://www-tapenade.inria.fr:8080/tapenade/paste.jsp, Download am 10.04.2012

[14]Der abgedruckte Quelltext wurde formatiert, um Platz zu sparen.

```
10      arg1d = _xd*_x + _x*_xd + _yd*_y + _y*_yd + _zd*_z + _z*_zd;
11      arg1  = _x*_x + _y*_y + _z*_z;
12      rd    = (arg1 == 0.0 ? 0.0 : arg1d/(2.0*sqrt(arg1)));
13      r     = sqrt(arg1);
14      rd    = (rd*r+r*rd)*r + r*r*rd;
15      r    *= r*r;
16      *xd   = *vxd;   *x = *vx;
17      *yd   = *vyd;   *y = *vy;
18      *zd   = *vzd;   *z = *vz;
19      *vxd  = (GME*_x*rd-GME*_xd*r)/(r*r);
20      *vx   = -GME*_x/r;
21      *vyd  = (GME*_y*rd-GME*_yd*r)/(r*r);
22      *vy   = -GME*_y/r;
23      *vzd  = (GME*_z*rd-GME*_zd*r)/(r*r);
24      *vz   = -GME*_z/r;
25 }
```

Listing 6.3: Implementierung der 3. Ableitung ($\frac{d^3\vec{r}}{dt^3}$) in der Programmiersprache C durch automatisches Differenzieren (Vorwärtsmodus)

Zusammenfassend bleiben folgende Punkte zu bedenken:

- Beim Laufzeitvergleich zwischen einem generierten und durch einen Programmierer implementierten Programm, ist - abhängig von der Erfahrung des Entwicklers - Letzteres oft schneller. Dies ist jedoch nur für relativ *einfache* Funktionen möglich, da sonst der zeitliche Aufwand für die Implementierung zu stark ansteigt. Bei *umfangreichen* Funktionen muss folglich auf generative Methoden zurückgegriffen werden.

- Werden die erarbeiteten Lösungsformeln der vorausgegangenen Unterpunkte verwendet, ist der Effekt der numerischen Auslöschung zu berücksichtigen.

- Bei symbolischer Differentiation werden die Gleichungen u.U. sehr umfangreich, was sich negativ auf die Programmlaufzeit auswirken kann. Außerdem steigt bei *sehr großen* Formeln die Wahrscheinlichkeit für Übertragungsfehler[15].

- Die automatische Differentiation mit SCT kann prinzipiell auf jede Programmiersprache angewendet werden. Somit kann auch *älterer* Quelltext[16] zum Einsatz kommen. Da die Ableitungsfunktion ausschließlich aus arithmetischen Basisoperationen aufgebaut ist, erzielen Optimierungen des Compilers gegenüber objektorientierter Operatorüberladung einen messbaren Geschwindigkeitsvorteil.

6.4 Fazit und weitere Arbeiten

Die hier verwendeten Methoden zur Bestimmung von höheren Ableitungen durch numerisches Differenzieren haben ihre Einsetzbarkeit gezeigt. Diese sollten jedoch nur

[15]Diese können beispielsweise auftreten, falls Formeln mit wxMaxima berechnet werden und diese in C++ implementiert werden sollen. Bestimmte Computeralgebrasysteme (z.B. MATHEMATICA) besitzen für diese Fälle entsprechende Exportfunktionen zu ausgewählten Programmiersprachen.

[16]In wissenschaftlichen Anwendungen sind oft ältere FORTRAN Dialekte (z.B. FORTRAN77) im Einsatz.

als Hilfsmittel zur Überprüfung erarbeiteter Lösungen verwendet werden. Außerdem sollten die Ergebnisse durch entsprechende Kontrollrechnungen verifiziert werden, da hierbei der Effekt der numerischen Auslöschung u.U. das Ergebnis ungewollt verschlechtern kann.
Ferner wäre es hilfreich, die Erstellung der Formeln zu automatisieren, da die Übertragung der Formeln aus der MATHEMATICA-Ausgabe in C++-Quelltext relativ aufwendig erscheint. Damit könnten Formeln für höhere Ableitungsterme je nach Bedarf dynamisch generiert werden. Eine alternative Möglichkeit, auf numerischem Weg höhere Ableitungen zu berechnen, sind Chebyshev-Polynome[17].
Weitere Ansätze zur Bestimmung höherer Ableitungen bieten die Methoden des automatischen Differenzierens (AD) oder der symbolischen Differentiation. Welche Methode letztendlich am besten geeignet ist, hängt von der Aufgabenstellung und den Zusatzbedingungen ab.

[17]Hierzu ist eine Implementierung in [Press07] zu finden. Den theoretischen Hintergrund findet man unter http://dlmf.nist.gov/3.11#ii (Download am 04.08.2011)

Kapitel 7

Toolbox : Entwickelte Programme und Bedienkonzepte

Schwerpunkt des Kapitels:

In diesem Abschnitt wird das Konzept einer **Toolbox** vorgestellt. Diese besteht unter Anderem aus Komponenten zur numerischen Lösung von gewöhnlichen Differentialgleichungen. Um die Bedienung derartiger Programme zu vereinfachen, wird im Folgenden ein minimaler Satz von Kommandozeilenoptionen erarbeitet. Deren Funktionsweise wird mit Beispielen gezeigt. Außerdem wird ein Programm zum Vergleich von *multiprecision*-Daten vorgestellt, was besonders beim Vergleich von *multiprecision*-Daten hilfreich ist.

7.1 Phasen der Datenverarbeitung

Die Verarbeitung von Daten kann in drei verschiedene Phasen unterteilt werden. Diese sind in Abbildung 7.1 dargestellt und beschreiben eine idealisierte Verarbeitungsreihenfolge.

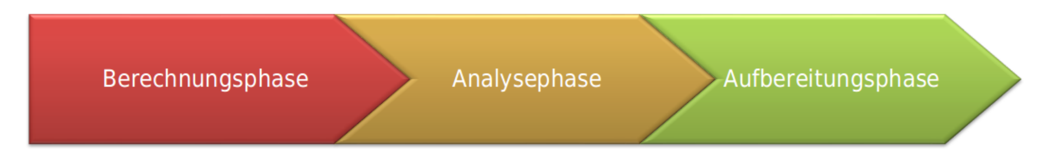


Abbildung 7.1: Phasen der Datenverarbeitung

- **Berechnungsphase**
  In der Berechnungsphase werden die Ergebnisse i.d.R. durch ein Computerprogramm erstellt. In den vorausgegangenen Kapiteln wurden fast ausschließlich die Methoden dieser Phase beschrieben. Hierzu zählen auch die verwendeten Bibliotheken zur Rechnung mit erhöhter Genauigkeit.

- **Analysephase**
  In dieser Phase werden die Daten, welche während der Berechnungsphase verarbeitet wurden, hinsichtlich bestimmter Kriterien analysiert. In den meisten Fälle ist es erforderlich, verschiedene Ergebnisse einander gegenüberzustellen und deren Genauigkeit zu beurteilen. Da diese Vergleiche meist auf das jeweils untersuchte Problem zugeschnitten sind, gibt es keine frei erhältlichen Analysewerkzeuge, die hierfür ohne Vorarbeit eingesetzt werden können.

- **Aufbereitungsphase**
  In der Aufbereitungsphase werden die Daten weiterverarbeitet, um das Ergebnis der Analysephase sichtbar zu machen. In den meisten Fällen wird diesbezüglich auf Programme wie *gnuplot*[1] oder *QtiPlot*[2] zurückgegriffen. Viele dieser Softwarepakete besitzen die Möglichkeit, Daten einzulesen und diese innerhalb der jeweiligen Programmumgebung zu analysieren. In diesem Zusammenhang ist Vorsicht geboten, da beispielsweise im Fall von *gnuplot* keine Möglichkeiten vorhanden sind, Zahlen mit mehrfacher Genauigkeit zu verarbeiten bzw. zu analysieren.

Als einleitendes Beispiel soll aufgezeigt werden, dass bei der Verarbeitung und Analyse von Gleitkommazahlen mit mehrfacher Genauigkeit die Ergebnisse exakt überprüft werden.

Beispiel 7.1 für „Verarbeitung von Gleitkommazahlen mit mehrfacher Genauigkeit“
In diesem Beispiel wurden zwei Lösungen des ungedämpften Duffing-Oszillators im Intervall zwischen 0 und 10 berechnet. Eine der beiden Lösungen wurde mit dem Potenzreihenintegrator[3] *unter Verwendung des bigfloat-Datentyps der LiDIA-Bibliothek*[4] *berechnet. Diese Lösung*

[1] siehe mehr dazu unter http://www.gnuplot.info/, Download am 05.08.2011
[2] siehe mehr dazu unter http://soft.proindependent.com/qtiplot.html, Download am 05.08.2011
[3] Bei jedem Integrationsschritt 40 Reihenterme berechnet.
[4] Die dezimale Genauigkeit wurde auf 100 signifikante Nachkommastellen eingestellt.

wird im Folgenden als $f_1(t)$ bezeichnet. Die andere Lösung wurde mit Hilfe der analytischen Lösung 30.Ordnung errechnet, welche ebenfalls den bigfloat-Datentyp *verwendet und folgend als $f_2(t)$ bezeichnet wird. Die beiden Ergebnisse sind tabellarisch in einer Datei (*Resultat.txt*) gespeichert und sollen nun analysiert werden. Die Daten sind durch Leerzeichen getrennt und liegen in folgender Reihenfolge zeilenweise gespeichert vor:*

- t_n $\quad f_1(t_n) \quad f_2(t_n)$

wobei n ein Platzhalter für die Zeilennummer ist.
Es soll nun gezeigt werden, wie mit Hilfe des Programms gnuplot *der Unterschied der beiden Lösungen durch Subtraktion der Tabellenwerte ermittelt wird. Das Ergebnis wird anschließend in einem Diagramm über der Zeit aufgetragen. Dies kann mit folgendem* gnuplot-*Befehl durchgeführt werden:*

- *plot 'Resultat.txt' using* $\underbrace{1 : (\$(2) - \$(3))}_{\text{spaltenweise Subtraktion der beiden Ergebnisse}}$

Das Resultat der Ausgabe ist in Abb. 7.2 dargestellt.

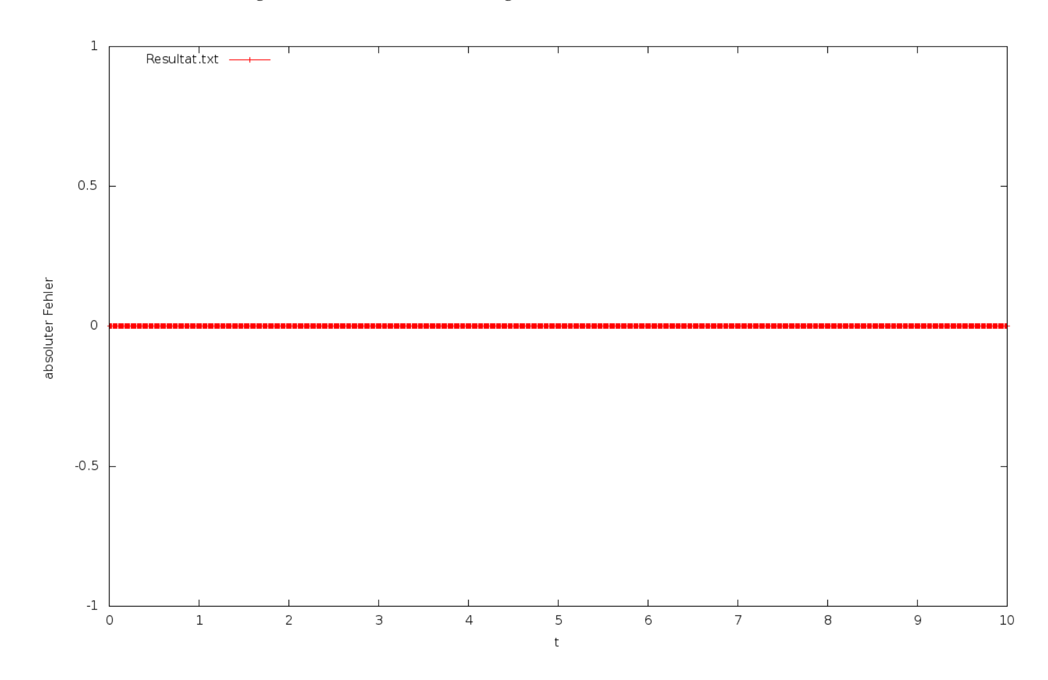


Abbildung 7.2: Auswirkungen unzureichender dezimaler Genauigkeit von gnuplot

Der tatsächliche Unterschied der beiden Lösungen ist in Abb. 7.3 dargestellt. Um dieses Resultat zu erhalten, wurden die beiden unterschiedlichen Lösungen mit Hilfe eines zusätzlichen Hilfsprogramms (s. Abschnitt 7.1.1), welches mit mehrfacher Genauigkeit rechnen kann, vorverarbeitet. Dieses Beispiel zeigt die Relevanz und den Bedarf an Werkzeugen dieser Art. Es existiert bereits das Programm numdiff[5] *auf Basis der GNU-MP-Bibliothek, welches den Unterschied zweier Dateien mit Hilfe beliebiger Genauigkeit ermitteln kann. In Fällen, in welchen die Daten in unterschiedlichen Dateien gespeichert sind, ist dies hilfreich. Jedoch müssen im vorliegenden Beispiel durch Duplizieren der Datei und anschließender Manipulation der Daten die Ergebnisse angepasst werden, was umständlich und fehleranfällig erscheint. Deshalb wurde das Programm* simple_mp_file_compare *entwickelt, welches in dieser Situation Abhilfe schafft (siehe dazu Abschnitt 7.1.1).*

[5]siehe mehr dazu unter http://www.nongnu.org/numdiff/, Download am 01.02.2011

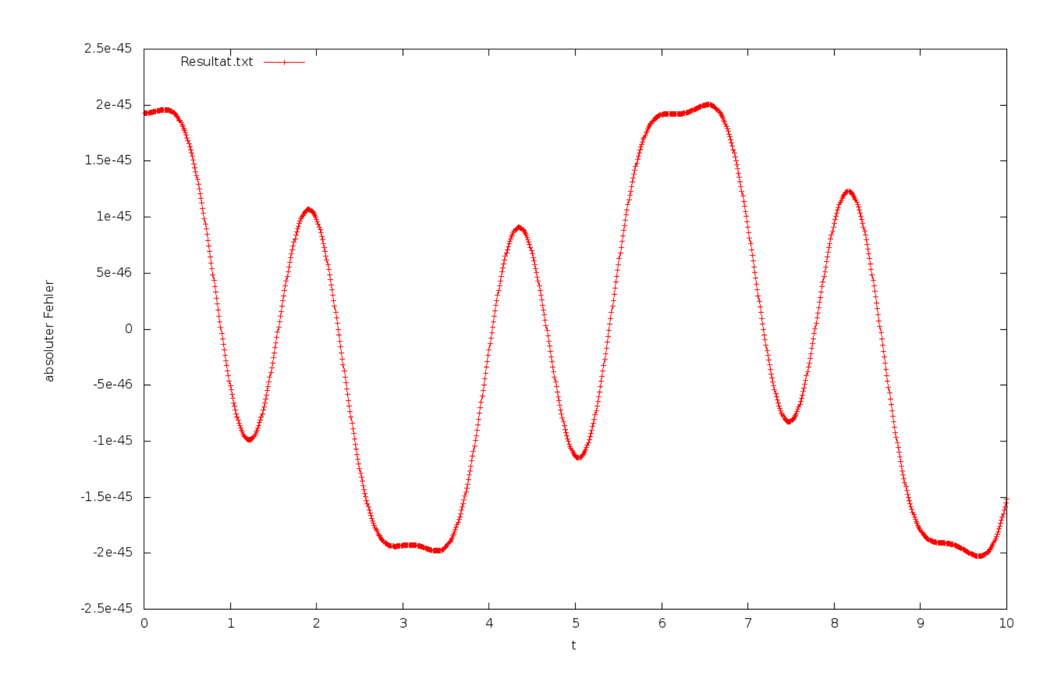


Abbildung 7.3: Unterschied der beiden berechneten Lösungen

7.1.1 Entwicklung eines Programms zur Vorverarbeitung von Daten mit hoher Genauigkeit

Wie im vorausgegangenen Abschnitt erwähnt, fehlt derzeit ein Programm zur Vorverarbeitung von Mess- bzw. Simulationsdaten, welche mit mehrfacher Genauigkeit erstellt wurden. Aus diesem Grund wurde für diesen Zweck ein derartiges Programmsystem[6] entwickelt. Hierzu sei kurz der Funktionsumfang des Programms dargestellt:

- Der Vergleich der Werte zweier ASCII[7]-Dateien, welche numerische Werte in Tabellenform speichern und durch Leerzeichen (oder Tabulatorzeichen) getrennt sind, wird ermöglicht.

- Außerdem ist es möglich, die Spalten innerhalb einer Datei auf identische Art und Weise zu vergleichen.

- Der Vergleich erfolgt durch Bildung des absoluten Fehlers[8] jeweils zweier Werte einer bestimmten Spalte (frei wählbar) der entsprechenden Datei.

- Die dezimale Genauigkeit zur Berechnung des absoluten Fehlers ist frei wählbar. Hierbei wird auf die LiDIA-Bibliothek[9] zurückgegriffen.

- Zusätzlich werden Kommentarzeilen in den zu verarbeiteten Dateien ignoriert. Das Symbol, welches einen Kommentar einleitet, ist frei wählbar.

[6]Das Programm trägt den Namen *simple_mp_file_compare*.

[7]Dies ist eine Zeichenkodierung, bei der jedem Zeichen ein Bitmuster von 7 Bit zugeordnet ist. Dies ist die amerikanische Version des Zeichensatzes. Das europäische Pendant hierzu ist ISO-646.

[8]definiert in Abschnitt 2.1.1

[9]siehe dazu [LiDIA]

- Ferner ist es möglich, zu jeder der verglichenen Spalten eine zusätzliche Spalte hinzugenerieren zu lassen, welche den kumulativen Fehler enthält.

In Abbildung 7.4 ist die Arbeitsweise von *simple_mp_file_compare* schematisch dargestellt. Hierbei wurde eine Anwendung gewählt, bei der zwei Dateien verwendet werden (file 1, file 2). Welche Spalten verarbeitet werden sollen, kann dem Programm in Form von Befehlen auf der Linux-Kommandozeile (options) mitgeteilt werden. Damit wird das Programm für die jeweilige Aufgabenstellung konfiguriert. In dem hier gewählten Beispiel soll aus der ersten Datei (file 1) die Spalte mit dem Index 0 in die Ausgabedatei übernommen werden. Außerdem soll aus der ersten Datei die Spalte mit dem Index 1 und aus der zweiten Datei die Spalte mit dem Index 2 verwendet werden, um den absoluten Fehler mitzubestimmen. Die dabei verwendete Genauigkeit, welche und wie viele Spalten verwendet werden, kann in den Optionen definiert werden.

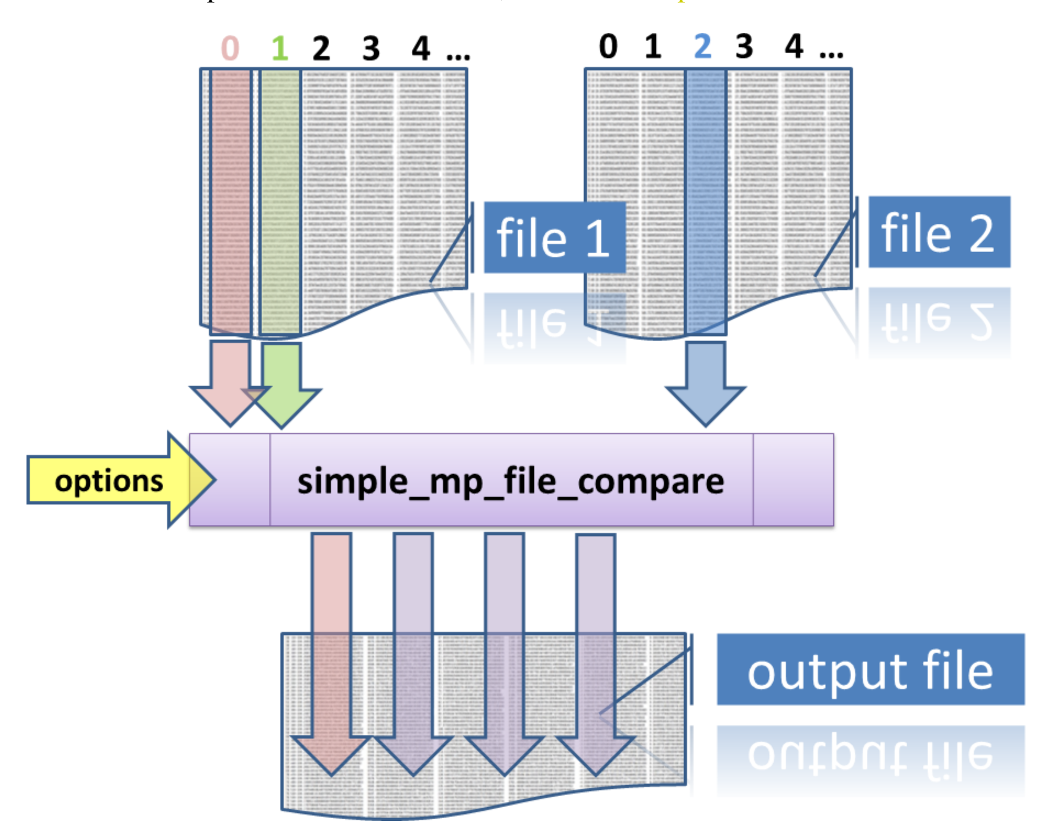


Abbildung 7.4: Das Arbeitsprinzip des Programms simple_mp_file_compare am Beispiel von zwei Eingabedateien

Beispiel 7.2 für „Berechnung des absoluten Fehlers mit simple_mp_file_compare“
Um die Leistungsfähigkeit von simple_mp_file_compare *unter Beweis zu stellen, wird das vorausgegangene Beispiel aufgegriffen. Darin wurden zwei Lösungen des ungedämpften Duffing-Oszillators im Intervall zwischen 0 und 10 berechnet. Eine der beiden Lösungen wurde mit dem Potenzreihenintegrator*[10] *unter Verwendung des* bigfloat*-Datentyps der LiDIA-Bibliothek*[11] *berechnet. Die zweite Lösung wurde mit Hilfe der analytischen Lösung 30.Ordnung errechnet,*

[10] Diesbezüglich werden bei jedem Integrationsschritt 40 Reihenterme berechnet.
[11] Die dezimale Genauigkeit wurde hierbei auf 100 signifikante Nachkommastellen eingestellt.

wofür ebenfalls der bigfloat*-Datentyp genutzt wurde. Die beiden Ergebnisse sind in einer Datei (Resultat.txt), tabellarisch abgespeichert. Sollen die Daten nun analysiert werden, so ist in vielen Fällen der absolute Fehler ein hilfreiches Kriterium. Hierzu wird* simple_mp_file_compare *verwendet, um aus der zweiten Spalte (Index 1) und dritten Spalte (Index 2) derselben Datei den absoluten Fehler aller Messwerte zu berechnen. Außerdem wird die erste Spalte (Index 0) in die Ausgabedatei (Output.txt) übernommen. Zusätzlich wird der kumulative absolute Fehler in die vierte Spalte (Index 3) der Ausgabedatei (Output.txt) eingefügt. Da die Ausgangsdaten auf 100 signifikante Dezimalstellen genau gerechnet werden, wird für die Weiterverarbeitung die interne Genauigkeit auf 200 Dezimalstellen eingestellt. Der benötigte Befehl auf der Linux-Kommandozeile ist in Listing 7.1 dargestellt. Zur Konfiguration besitzt* simple_mp_file_compare *ein CLI . Die wichtigsten*[12] *Optionen werden hier kurz erklärt:*

[CLI] Command Line Interface

- *$-f1$ =[Dateiname],$-f2$ =[Dateiname]*
  Mit dieser Option können die Eingabedateien festgelegt werden. Falls nur eine Datei verwendet werden soll und darin einige Spalten verarbeitet werden sollen, ist für beide Optionen derselbe Dateiname anzugeben.
- *$-k1$ =[Spaltenindex],$-k2$ =[Spaltenindex]*
  Hier können die Spaltenindizes der jeweiligen Dateien angegeben werden, welche in die Ausgabedatei mit übernommen werden. Sollen mehrere Spalten aus einer Datei übernommen werden, dann sind die Indizes mit Kommas zu trennen.
- *$-c1$ =[Spaltenindex],$-c2$ =[Spaltenindex]*
  Durch Angabe der Spaltenindizes können die Spalten bestimmt werden, aus denen der absolute Fehler gerechnet wird. Mehrere Spaltenindizes werden durch Komma getrennt angegeben. Bei dieser Option ist die gleiche Anzahl der Spaltenindizes für beide Optionen erforderlich, andernfalls bricht das Programm mit einer Fehlermeldung ab.
- *–append-cumulative-col*
  Wird diese Option verwendet, dann wird neben jeder der Spalten, bei denen ein absoluter Fehler berechnet wird, eine zusätzliche Spalte generiert. Diese enthält die Werte des absoluten, kumulativen Fehlers.
- *$-l$[Multiprecision-Bibliothek]*
  Mit dieser Option kann die zur Berechnung verwendete Multiprecision-Bibliothek ausgewählt werden. In diesem Beispiel wurde die LiDIA-Bibliothek ausgewählt.
- *$-p$[Anzahl Dezimalstellen]*
  Diese Option funktioniert nur in Verbindung mit der Option zur Definition einer bestimmten Multiprecision-Bibliothek ($-l$). Damit kann die Anzahl signifikanter Nachkommastellen, welche man zur Berechnung verwendet, angegeben werden.
- *$-o$ =[Name der Ausgabedatei]*
  Hiermit kann der Name bzw. der Pfad der Ausgabedatei angegeben werden.

[12] Eine vollständige Liste aller Optionen (inkl. Beispiel) kann mit dem Linux-Kommandozeilenbefehl ./simple_mp_file_compare -h angezeigt werden.

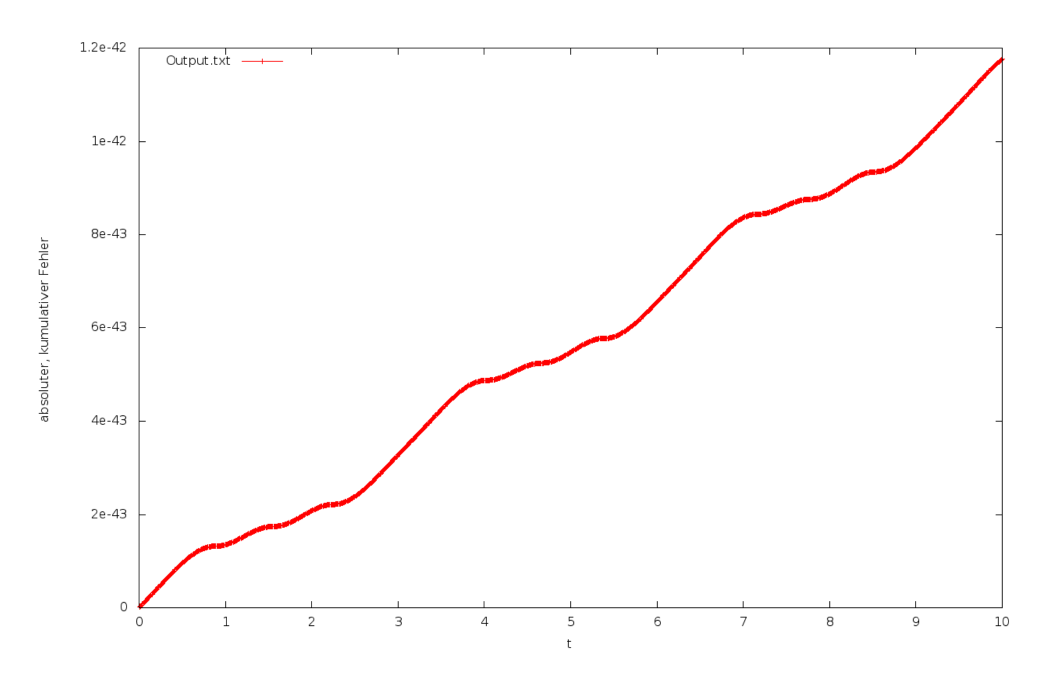


Abbildung 7.5: Absoluter kumulativer Fehler zweier Lösungen der ungedämpften Duffing-Oszillatorgleichung

```
1 ./simple_mp_file_compare -f1=Result.txt -f2=Result.txt
2                          -k1=0 -c1=1 -c2=2 --append-cumulative-col
3                          -lLiDIA -p200 -o=Output.txt
```

Listing 7.1: Aufruf von simple_mp_file_compare

7.1.2 Weiterentwicklung von simple_mp_file_compare

Um den Vergleich von Dateien und somit die Verwendung von simple_mp_file_compare benutzerfreundlicher zu gestalten, wurde ein Konzept einer graphischen Benutzeroberfläche entworfen. Das GUI basiert auf der *wxWidgets*[13]-Programmbibliothek. Dies ist eine *Open-Source*-Programmbibliothek, mit der es möglich ist, ein Programm auf einer bestimmten Plattform (z.B. Linux) zu entwickeln und es später auf einer anderen Betriebssystemplattform (z.B. Windows) zu verwenden. Das Design wurde mit dem RAD-Programm *wxFormBuilder*[14] erstellt. In Abbildung 7.6 ist das entworfene Design dargestellt.
Im Folgenden werden die Grundfunktionen des Designentwurfs erläutert:

[GUI] Graphical User Interface

[RAD] Rapid Application Development

[13] siehe mehr dazu auf der Webseite des Projekts: http://www.wxwidgets.org, Download am 13.10.2011
[14] siehe mehr dazu auf der Webseite des Projekts: http://www.wxformbuilder.org, Download am 13.10.2011

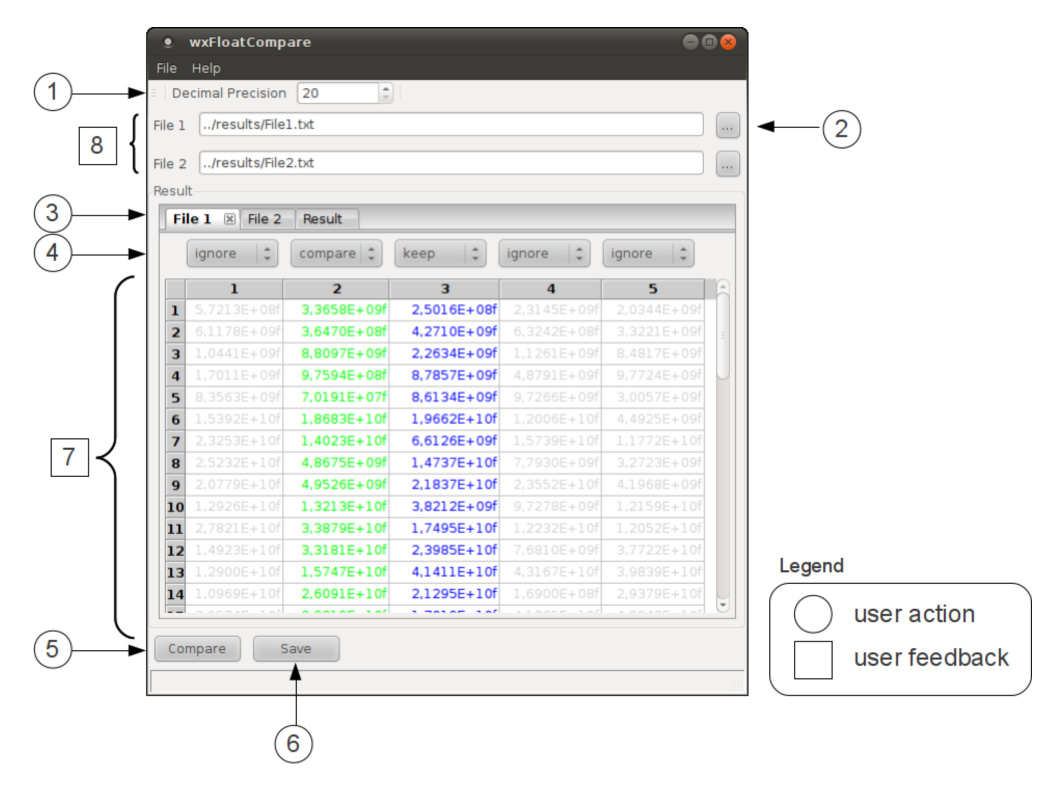


Abbildung 7.6: Grafische Benutzeroberfläche von wxFloatCompare

1. Hier kann der Benutzer die Anzahl signifikanter Nachkommastellen eingeben, welche zur Berechnung verwendet werden.

2. Drückt der Benutzer auf einen der beiden Buttons, so kann die jeweilige Eingabedatei geladen werden. Der anschließend verwendete Pfad wird links daneben angezeigt. Dort ist es außerdem möglich, den Pfad direkt einzugeben.

3. In den ersten zwei Reitern wird der Inhalt der jeweils geladenen Dateien dargestellt. Diese tragen die Beschriftung *File 1* und *File 2*. Im dritten Reiter werden die Ergebnisse nach der Verarbeitung angezeigt.

4. In dieser Zeile erhält der Benutzer zu jeder Zeile einer geladenen Datei die Möglichkeit zwischen drei Optionen auszuwählen:

   - *ignore:* Diese Spalte bei soll bei der Verarbeitung ignoriert werden. Hat der Benutzer diese Option gewählt, dann werden die Werte in hellgrau eingefärbt.
   - *keep:* Die Werte dieser Spalte werden in die Ausgabe übernommen und nicht zur Verarbeitung verwendet. Ist diese Option eingestellt, so werden die Werte dieser Spalte in blau eingefärbt.
   - *compare:* Mit dieser Option wird eine Spalte zum Vergleich ausgewählt. Ist diese Option aktiv, so werden die Werte dieser Spalte grün eingefärbt.

5. Hiermit kann der Benutzer die Berechnung beginnen, nachdem er die jeweiligen Spalten zum Vergleich ausgewählt hat.

6. Ist die Berechnung abgeschlossen, dann können die Ergebnisse in einer Datei gespeichert werden.

7. In dieser Anzeige erhält der Benutzer eine Rückmeldung des Programms, welche Daten geladen sind und welche Spalten er momentan ausgewählt hat.

8. Hier wird dem Benutzer angezeigt, welche Dateien bzw. Dateipfade gerade geladen sind.

7.2 Programme zur Lösung gewöhnlicher Differentialgleichungen

Im Rahmen dieser Arbeit wurden eine ganze Reihe verschiedener Aufgabenstellungen untersucht. Zu diesem Zweck wurden speziell dafür angepasste Programme entwickelt, welche auf die jeweils geforderte Aufgabenstellung zugeschnitten sind. Um diese Programme zu bedienen gibt es i.d.R. eine Vielzahl verschiedener Konfigurationsmöglichkeiten. Im folgenden Abschnitt werden diese kategorisiert und anschließend definiert. Jedes der entwickelten Programme besitzt somit einen beschränkten Satz an Basisoptionen. Dadurch wird die Nutzerfreundlichkeit der jeweiligen Programme entscheidend vereinfacht und der Funktionsumfang auf ein erträgliches Minimum gesenkt. Ferner werden die jeweils verwendeten Kommandozeilenbefehle an den GNU-Standard[15] für Kommandozeilenbefehle angelehnt.

7.2.1 Kategorisierung der Einstellungsmöglichkeiten eines Programms zur Lösung von Bewegungsproblemen

Hier sollen zunächst die Eingabeoptionen eines allgemeinen Programms zur Lösung eines nicht näher spezifizierten Bewegungsproblems betrachtet werden. Eine Submenge der hierfür benötigten Eingabeoptionen ist stichpunktartig in Abb. 7.7 abgebildet.

Diese können in folgende Hauptkategorien unterteilt werden:

- Numerische Einstellungen
  Diese beziehen sich auf den jeweils zur Berechnung verwendeten Datentyp. Entscheidet man sich für einen bestimmten Datentyp, bei dem beispielsweise die dezimale Genauigkeit frei wählbar ist, so muss diese ebenfalls angegeben werden, d.h. diese Option hängt von der anderen ab. Wird andererseits ein Basisdatentyp (z.B. *double*) verwendet, bei dem die Genauigkeit vorgegeben ist, dann ist diese Option nutzlos.

- Integrationseinstellungen
  Bei den Einstellungen zur Integration können sowohl die Start, als auch die Endepoche angegeben werden. Außerdem kann die Schrittweite der Integration und deren Anzahl festgelegt werden.

- Integratorauswahl und Konfiguration
  Bei dieser Kategorie kann der zur Berechnung verwendete Algorithmus gewählt

[15] siehe mehr dazu in der Onlinedokumentation unter http://www.delorie.com/gnu/docs/GNU/standards_18.html, Download am 14.10.2011

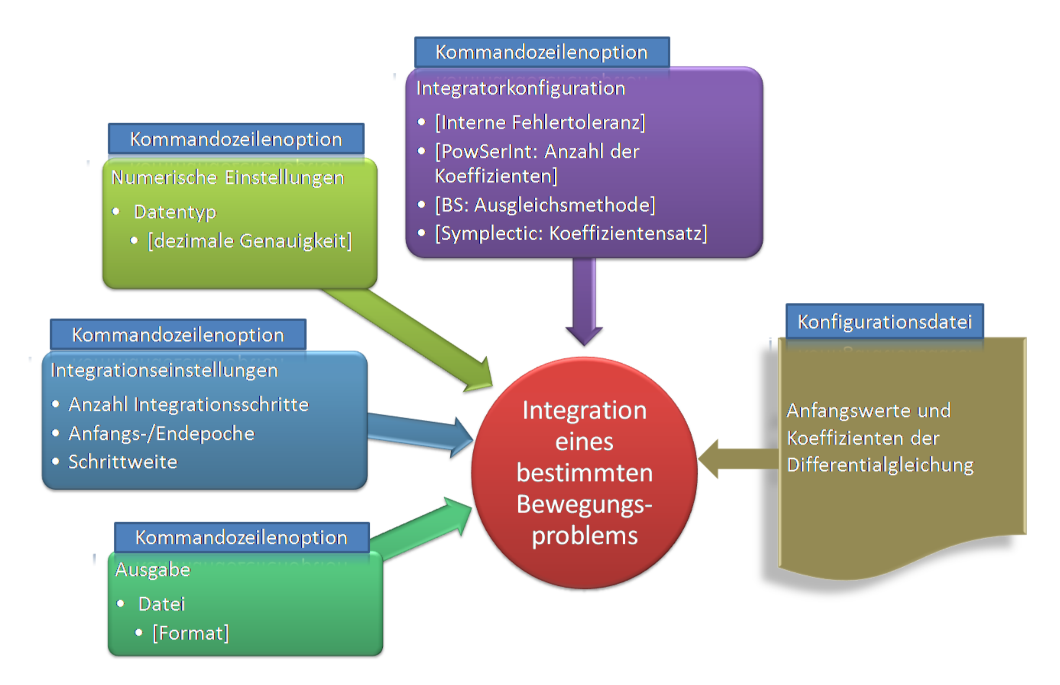


Abbildung 7.7: Einstellungsmöglichkeiten eines Programms zur Lösung von Bewegungsproblemen

werden. Ist ein bestimmter Algorithmus ausgewählt, so können i.d.R. zusätzliche Einstellungen vorgenommen werden. Wird beispielsweise der Potenzreihenintegrator (PowSerInt) verwendet, dann muss die Anzahl der Potenzreihenkoeffizienten angegeben werden. Folglich werden auch hier Optionen benötigt, welche von den anderen implizit abhängig sind.

- Ausgabe
  Hierbei kann die Zieldatei und das jeweilige Ausgabeformat angegeben werden.

- Anfangswerte und Parameter der Differentialgleichung
  Aus einer Konfigurationsdatei werden sowohl die Anfangswerte zur Integration, als auch die Parameter der jeweiligen Differentialgleichung beim Programmstart abgelesen.

Somit können folgende Kommandozeilenoptionen definiert werden:

- *–config-file=[Pfad zur Konfigurationsdatei]*
  Durch Angabe der Konfigurationsdatei werden die Anfangswerte und die Parameter der Differentialgleichung vor Beginn der Rechnung in das Programm geladen. Es wäre prinzipiell möglich, Angaben ebenfalls via Kommandozeilenoptionen in das Programm einzuspeisen. Hier erschient es praktikabel, diese Parameter in einer Datei abzulegen, da in den meisten Fällen eine Aufgabenstellung mit unterschiedlichen numerischen Genauigkeiten bzw. Algorithmen untersucht wurde. Beispielhaft ist in Listing 7.2 die Konfigurationsdatei zur Berechnung des Duffing-Oszillators angegeben. Das hierfür verwendete Datenformat ist XML-ähnlich und benötigt zum Lesen der Datei einen speziell dafür ent-

[XML] EXtensible Markup Language

wickelten Konfigurationsparser[16]. Im Allgemeinen ist die Konfigurationsdatei in zwei Abschnitte unterteilt. Im Abschnitt *InitialValues* werden die Anfangswerte zur Integration festgelegt. Dabei bezeichnet der Parameter x den Anfangsort und der Parameter vx die Anfangsgeschwindigkeit. Im Abschnitt *Parameter* werden die Konstanten, welche die Bewegung des Duffing-Oszillators beschreiben, definiert.

```
1  <ODE>
2      <InitialValues>
3          Name  =  Experiment1
4          x     =  1
5          vx    =  0
6      </InitialValues>
7      <Parameter>
8          Delta   =  1.0e-2
9          Omega   =  1
10         Damped  =  false
11         Gamma   =  1.0e-2
12     </Parameter>
13 </ODE>
```

Listing 7.2: Konfigurationsdatei für den Duffing-Oszillator (*Duffing.conf*)

- –float-type=*[Name des Datentyps]*
  Mit dieser Option kann der zur Berechnung verwendete Datentyp festgelegt werden. Wird ein Datentyp mit frei wählbarer Anzahl signifikanter Dezimalstellen verwendet, dann ist es erforderlich, diese mit Hilfe der Option –precision= festzulegen.

- –precision=*[Anzahl signifikanter Nachkommastellen]*
  Mit dieser Option kann die Anzahl der signifikanten Dezimalstellen festgelegt werden. Diese ist jedoch nur bei Datentypen gültig, bei denen die Genauigkeit eingestellt werden kann, d.h. bei den primitiven Datentypen *float*, *double* und *long double* hat diese Option keine Wirkung.

- –integration-stepnumber=*[Anzahl der Integrationsschritte]*
  Die Anzahl der Integrationsschritte kann mit dieser Option festgelegt werden. Hierbei ist darauf zu achten, dass Integerzahlen verwendet werden.

- –integration-stepsize=*[Integrationsschrittweite]*
  Hier kann die Integrationsschrittweite definiert werden. Ist diese beispielsweise auf den Wert 1.5 eingestellt (–integration-stepsize=1.5), dann erfolgt die Integration in 1.5-er Schritten.

- –integration-start=*[Startzeitpunkt (Epoche)]*
  Die Startepoche der Integration kann mit Hilfe dieser Option festgelegt werden.

[16] Dieser wurde am Geodätischen Observatorium in Wettzell entwickelt. Mehr Informationen zu diesen und anderen Komponenten dieser Art sind auf folgender Webseite zu finden: http://econtrol-software.de/

Dabei ist zu beachten, dass der Zeitpunkt, zu dem die Integration beendet wird (t_{end}), wie folgt berechnet werden kann:

$$t_{end} = t_{start} + n * h \tag{7.2.1}$$

Wobei n ein Platzhalter für die Anzahl der Integrationsschritte und h ein Platzhalter für die Integrationsschrittweite ist.

- *–integrator-type=[Name des Integrators]*
  Die Differentialgleichung kann mit unterschiedlichen Lösungsverfahren gelöst werden. Hier kann eines der folgenden Lösungsverfahren ausgewählt werden:
  - **BS**:
    Burlisch-Stoer-Verfahren
  - **RK2,RK3,RK4,RK8,RK10**:
    Runge-Kutta-Verfahren unterschiedlicher Ordnung
  - **MMID**:
    Modified-Midpoint Methode
  - **GJ4**:
    Gauss-Jackson (4.Ordnung)
  - **DOPRI**:
    Dormand–Prince Methode
  - **SG**:
    Shampine Gordon - Verfahren
  - **POWSERINT**:
    Potenzreihenintegrationsverfahren
  - **SYMP4,SYMP6,SYMP8**:
    Symplektische Verfahren unterschiedlicher Ordnung
  - **LFROG**:
    Leapfrog-Verfahren
- *–integrator-abstol=[Absolute Fehlertoleranz]*
  Wird ein Verfahren mit adaptiver Schrittweitensteuerung verwendet, so ist außerdem die Angabe einer absoluten Fehlertoleranz nötig. Mit dieser Option kann diese dem Programm mitgeteilt werden.
- *–integrator-coeffs=[Anzahl Potenzreihenkoeffizienten]*
  Wird der Potenzreihenintegrator ausgewählt, dann kann mit dieser Option die Anzahl der verwendeten Potenzreihenkoeffizienten festgelegt werden.
- *–output-file=[Pfad zur Ausgabedatei]*
  Soll das Ergebnis in eine bestimmte Textdatei geschrieben werden, dann kann diese Option verwendet werden. Wird auf diese verzichtet, dann wird das Ergebnis auf die Standardausgabe geschrieben.

Beispiel 7.3 für „Integration des Lorenz-Attraktors“
Hier soll beispielhaft der Lorenz-Attraktor[17] *mit Hilfe des Burlisch-Stoer-Integrators gelöst werden. Die Lorenz-Gleichung besteht aus drei gekoppelten, nichtlinearen Differentialgleichungen der Form:*

$$\begin{aligned} \frac{dx}{dt} &= a(y-x) \\ \frac{dy}{dt} &= x(b-z)-y \\ \frac{dz}{dt} &= xy-cz \end{aligned} \tag{7.2.2}$$

Wobei die Variablen a, b und c als Konstanten betrachtet werden. Zur Integration werden die Werte $a = 10$,$b = 28$ und $c = \frac{8}{3}$ gesetzt. Als Anfangswerte werden die Werte $x = 10$,$y = 20$ und $z = 30$ verwendet. Um diese Aufgabenstellung zu lösen, wird auf die oben definierten Kommandozeilenparameter zurückgegriffen und das Ergebnis in der Datei LorenzBS.txt *gespeichert.*
*Somit muss das Programm (*IntegrateLorenz*) mit bestimmten Kommandozeilenparametern aufgerufen werden. Diese sind in Listing 7.3 angegeben. In Zeile 1 wird erst das Programm aufgerufen und dann die Konfigurationsdatei*[18] *angegeben. Diese enthält die Anfangswerte und die Konstanten zur Berechnung des Lorenz-Attraktors.*

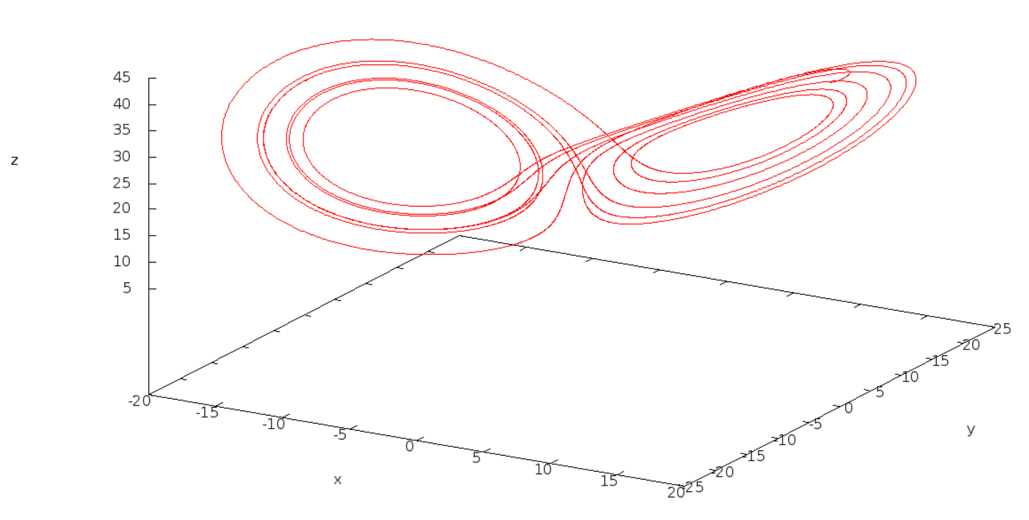


Abbildung 7.8: Graphische Darstellung des Lorenz-Attraktors

[17] Mehr Informationen findet man dazu unter [Schuster05] (ab Seite 109).
[18] Die Datei *Lorenz.conf* ist im Anhang F.1 auf Seite 217 zu finden.

```
1 ../bin/IntegrateLorenz   --config-file=../conf/Lorenz.conf
2                          --float-type=Xdouble
3                          --integrator-type=BS
4                          --integrator-abstol=1.0e-15
5                          --integration-start=0
6                          --integration-stepsize=0.001
7                          --integration-stepnumber=10000
8                          --output-file=../samples/LorenzBS.txt
```

Listing 7.3: Kommandozeilenparameter zur Lösung des Lorenz-Attraktors

7.2.2 Anmerkung zum Lorenz-Attraktor

Die Lorenz'schen Differentialgleichungen reagieren auffällig sensibel auf Veränderungen. Sogar das Umstellen der Gleichung durch einfaches Ausklammern der Konstanten a auf der rechten Seite von

$$\frac{dx}{dt} = ay - ax \tag{7.2.3}$$

nach

$$\frac{dx}{dt} = a(y - x) \tag{7.2.4}$$

hat bereits einen Einfluss auf das Endergebnis. Die beiden Ausdrücke sind mathematisch identisch, jedoch werden sie aufgrund des Rechengesetzes (*Punkt vor Strich*) anders abgearbeitet. Infolgedessen kann ein anderer Rundungsfehler auftreten, der das Endergebnis verändert. Bei derart eng gekoppelten Differentialgleichungen ist folglich Vorsicht geboten. Andererseits kann mit diesen Gleichungen chaotisches Verhalten nachgewiesen werden und ist folglich für die mathematische Chaostheorie nützlich.

7.3 Fazit und weitere Entwicklungen

In diesem Abschnitt wurde ein Bedienkonzept für die Programme zur Lösung von Bewegungsproblemen auf Basis von Kommandozeilenoptionen erarbeitet. Dieser funktionelle Ansatz könnte mit einer graphischen Benutzeroberfläche deutlich komfortabler gestaltet werden. Dazu könnte, wie schon bei *wxFloatCompare*, die plattformunabhängige Klassenbibliothek wxWidgets[19] oder Qt[20] verwendet werden. Entscheidend bei einer derartigen Entwicklung ist, dass aus Sicht der Informatik die Trennung zwischen eigentlicher **Berechnung** und **GUI** gewahrt bleibt, da sonst die Wartbarkeit des Quellcodes mit zunehmender Funktionalität nicht mehr gewährleistet ist. Außerdem wäre dadurch eine Client/Server Architektur möglich, bei der die graphische Benutzeroberfläche nicht zwingend auf demselben Rechner betrieben werden muss[21]. Dies kann durch Spezifikation von Schnittstellenfunktionen gewährleistet werden, was i.d.R. mit Hilfe einer IDL[22] realisiert wird. Hierbei werden die Schnittstellenfunktionen in abstrakter Form definiert und einem Softwaregenerator übergeben, der für

[GUI] Graphical User Interface

[IDL] Interface Description Language

[19] Siehe mehr dazu unter http://wxwidgets.org, Download am 19.10.2011
[20] Siehe mehr dazu unter http://qt.nokia.com/products/, Download am 19.10.2011
[21] Siehe mehr dazu unter [Neidhardt08]
[22] Bekannte Repräsentanten hierfür sind CORBA und SOAP.

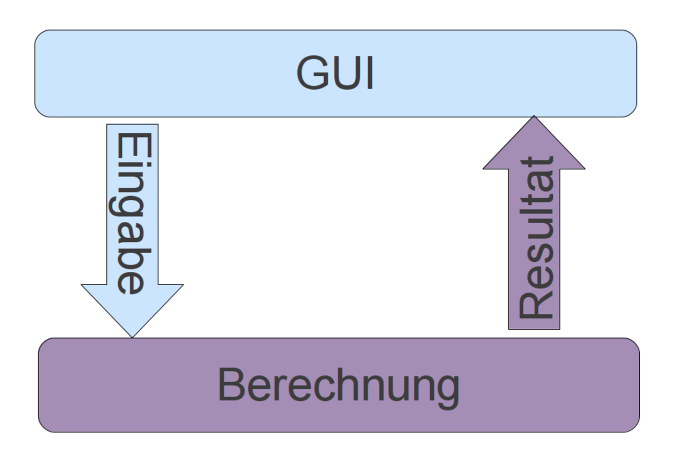


Abbildung 7.9: Unterteilung zwischen Anwendung und Darstellung

Client und Server die jeweiligen Funktionen generiert. Werden ausschließlich diese Funktionen zur Kommunikation zwischen den beiden Teilen verwendet, so ist diese Trennung gewährleistet. Darüber hinaus können die beiden Programmteile (**GUI** und **Berechnung**) von unterschiedlichen Personen betreut werden, was die Weiterentwicklung längerfristig fördern würde.
Außerdem wäre es hilfreich, die numerischen Ergebnisse direkt auf graphischem Weg auswerten zu können. Aktuell werden die errechneten Daten erst in eine ASCII-Datei geschrieben und anschließend mit einem Hilfsprogramm (z.B. *gnuplot*) visualisiert. Hier könnte das C++-Interface zum *gnuplot*-Programm (*gnuplot-cpp*[23]) einen nützlichen Beitrag leisten. Dies ist eine Erweiterung zum *gnuplot*-Programm mit direkter C++-Sprachunterstützung. Damit können, ohne den Umweg über ASCII-Dateien, die jeweiligen Daten direkt graphisch visualisiert und bei Bedarf in ein Bild exportiert werden.

[23] siehe mehr dazu unter http://code.google.com/p/gnuplot-cpp/, Download am 19.10.2011

Kapitel 8

Anwendungsbeispiele zur Integration von Bewegungsproblemen

Schwerpunkt des Kapitels:

Aufgrund der immer größer werdenden Ansprüche hinsichtlich der Genauigkeit bei der Beobachtung von Laserentfernungsmessungen zu Satelliten und zum Mond, werden konsequenterweise die Anforderungen an die numerischen Verfahren ebenfalls erhöht. In diesem Abschnitt sollen sowohl die Leistungsfähigkeit, als auch die Schwächen heutiger Verfahren exemplarisch aufgezeigt werden. Hierzu werden zwei aus der Praxis bekannte Anwendungsfälle aufgegriffen und das bei der numerischen Berechnung anfallende Fehlerbudget betrachtet. Das jeweilig ermittelte Fehlerbudget gilt als Qualitätskriterium zum Vergleich der unterschiedlichen Berechnungsmethoden.

- Im ersten Anwendungsfall wird die rechnerische Genauigkeit zweier sich verfolgender Satelliten, wie es beispielsweise bei der GRACE-Mission der Fall ist, mit verschiedenen Methoden simuliert und untersucht.
- Im zweiten praktischen Beispiel wird mit Hilfe des Programms *EPHEMER* eine Ephemeride der geozentrischen Mondbahn über einen Zeitraum von 32 Jahre hinweg berechnet und auf ihre Genauigkeit untersucht.

8.1 Berechnung der Abstände zwischen GRACE-A und -B Satelliten

In diesem Abschnitt sollen die Satellitenbahnen der GRACE-Mission simuliert werden. Hierzu werden verschiedene numerische Integrationsverfahren eingesetzt. Die jeweiligen Berechnungen werden zunächst mit herkömmlicher, dezimaler Genauigkeit und anschließend mit erhöhter Genauigkeit durchgeführt.

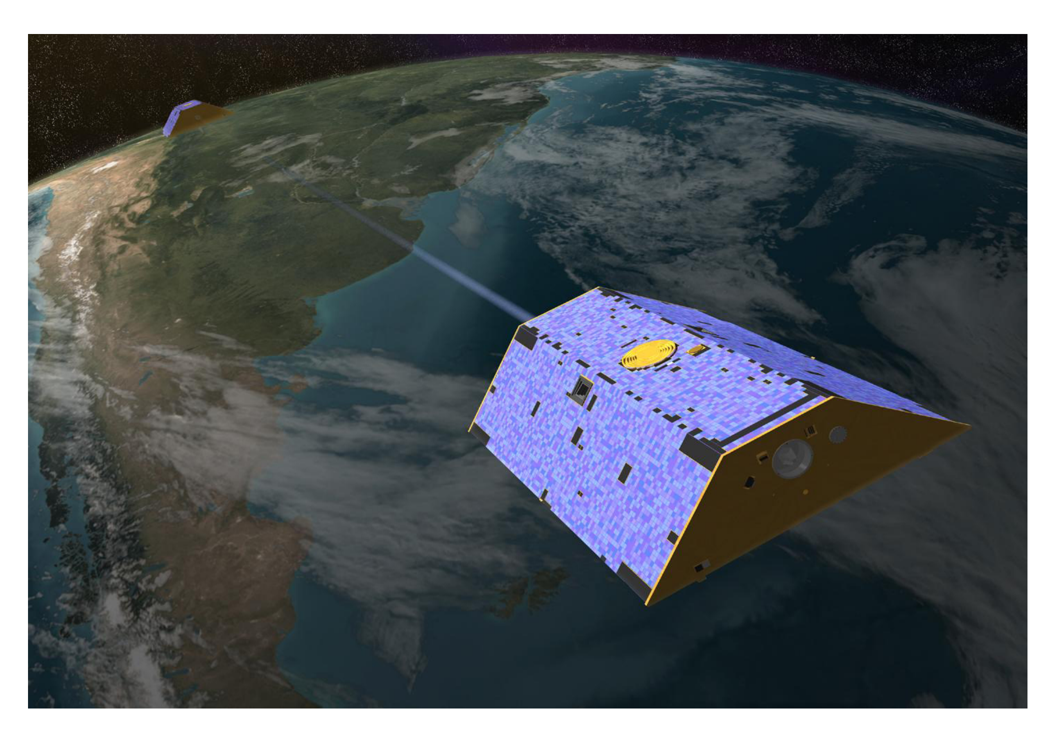


Abbildung 8.1: GRACE-A und B Satelliten im Erdorbit

Mit Hilfe dieser Vorgehensweise können numerische Effekte bzw. das Fehlerbudget der jeweils ungenaueren Berechnung sichtbar gemacht werden.

8.1.1 Die GRACE-A und B Mission

Der Name GRACE ist ein Akronym, das für *Gravity Recovery And Climate Experiment* steht. Diese deutsch-amerikanische Satellitenmission hat das Ziel, das Gravitationsfeld der Erde in hoher räumlicher und zeitlicher Auflösung zu bestimmen. Die Mission besteht aus zwei identischen Satelliten, welche sich in einem polaren Orbit auf einer Höhe von 500 Kilometer mit einem Abstand von 230 Kilometer verfolgen. Abb. 8.1[1] illustriert das GRACE-Satellitenpaar.
Die beiden Satelliten bestimmen mit Hilfe eines an Bord befindlichen GPS-Empfängers ihre momentane Position. Ferner wird kontinuierlich der Abstand der beiden Satelliten durch ein Mikrowellenverfahren bestimmt. Anhand der gesammelten Information ist es möglich, ein Modell des Gravitationsfeldes der Erde zu bestimmen. Mehr Informa-

[1] Dieses Bild wurde der Webseite der GRACE Mission entliehen http://www.csr.utexas.edu/grace/, Download am 03.05.2010.

tionen zu diesem Modell und der GRACE-Mission findet man auf der Webseite zur Mission[2].

8.1.2 Anfangswerte zur Simulation

Um die Mission so genau wie möglich in der Simulation nachzustellen, wurden die Keplerschen Bahnelemente in kartesische Startkoordinaten umgerechnet. Diese sind in herkömmlicher, dezimaler Genauigkeit in Tabelle 8.1 aufgelistet. Für die Berechnung mit höherer Genauigkeit wurden entsprechend genauere Anfangswerte verwendet.

Komponente	**GRACE-A**	**GRACE-B**	**Einheit**
$x(t_0)$	6771.358863	6767.4414785087768080	[km]
$y(t_0)$	-0.1458356217974367E-94	8.1467152155126077979	[km]
$z(t_0)$	-0.4119020590424369E-93	230.09819572516469395	[km]
$\dot{x}(t_0)$	0.46723802140296940E-96	-0.261133431327396422	[km/s]
$\dot{y}(t_0)$	0.27160990850338429	0.2714520466861765550	[km/s]
$\dot{z}(t_0)$	7.67142384457644789	7.6669722925044407280	[km/s]

Tabelle 8.1: Anfangswerte von GRACE-A und B

Eine kurze Kontrollrechnung soll zeigen, dass die hier verwendeten Anfangswerte zwei Satelliten beschreiben, welche sich in einem Abstand von ca. 230 km verfolgen. Um den Abstand der beiden Satelliten $d_{A,B}(t)$ zu einem bestimmten Zeitpunkt t zu berechnen, ergibt sich folgender Zusammenhang:

$$d_{A,B}(t) = \sqrt{(x_A(t) - x_B(t))^2 + (y_A(t) - y_B(t))^2 + (z_A(t) - z_B(t))^2} \quad (8.1.1)$$

Hierbei ist $x_A(t)$ ein Platzhalter für die kartesische X-Koordinate des GRACE-A-Satelliten zum Zeitpunkt t. Das Einsetzen der Anfangswerte aus Tabelle 8.1 ergibt folgendes Ergebnis:

$$d_{A,B}(t) = 230.275692477584215112074161879718303680 4...[km] \quad (8.1.2)$$

8.1.3 Simulationen

Um die Abstände der beiden GRACE-Satelliten-Orbits zu berechnen, werden diese mit unterschiedlichen Berechnungsverfahren ermittelt und die jeweiligen Ergebnisse einander gegenübergestellt. Ferner wurde die Anzahl der signifikanten Nachkommastellen variiert, um numerische Effekte der jeweils untersuchten Verfahren sichtbar zu machen. In Abbildung 8.2 sind die Abstände zwischen GRACE-A und -B in einem Zeitraum von fünf Stunden aufgetragen.

[2] http://www.csr.utexas.edu/grace/, Download am 10.10.2011

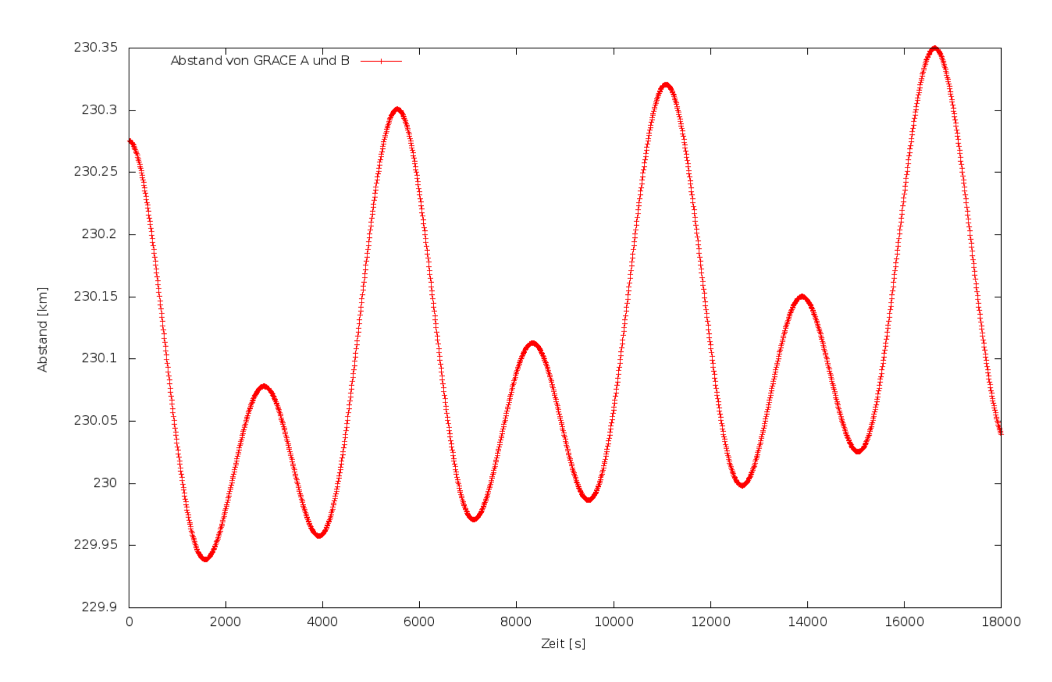


Abbildung 8.2: Abstände der GRACE-A und -B-Satelliten über einen Zeitraum von fünf Stunden

Hierbei fällt auf, dass die Abstände der beiden Satelliten, gemessen von *peak to peak*, im Laufe der Zeit als Folge der gewählten Anfangsbedingungen zunehmen. Der maximale Abstand lag bei 230.349882 Kilometer und der minimale Abstand bei 229.938827 Kilometer.
Als anisotropes Gravitationsfeld wurde das EGM96-Modell[3] mit Grad und Ordnung 40 verwendet. Durch den Vergleich der Einzellösungen, welche mit unterschiedlichen dezimalen Genauigkeiten berechnet werden, können u.U. auftretende, numerische Ungenauigkeiten sichtbar gemacht werden. Dabei entsteht beim Vergleich der unterschiedlichen Lösungen eine individuelle Fehlerkurve. Aus dieser kann sowohl der Fehlerverlauf, als auch die verbleibende Restgenauigkeit des Ergebnisses abgelesen werden.

8.1.4 Abstandsdifferenzen

Um die unterschiedlich ermittelten Abstandslösungen $d_{A,B}$ miteinander vergleichen zu können, werden Abstandsdifferenzen gebildet. Diese sind definiert:

$$d_{A,B}^{I1,I2,P1,P2}(t) = |d_{A,B}^{I1,P1}(t) - d_{A,B}^{I2,P2}(t)| \tag{8.1.3}$$

wobei die hochgestellten Indizes bedeuten:

- $I1, I2$
  Bezeichnung des jeweils verwendeten Integrators.

- $P1, P2$
  Dezimale Genauigkeit, die zur Berechnung verwendet wurde.

[3] http://cddis.nasa.gov/926/egm96/egm96.html, Download am 08.10.2011

Somit kann ein bestimmter Abstand, welcher mit dem Runge-Kutta-Verfahren (10.Ordnung) bestimmt wurde, unter Verwendung eines bestimmten Datentyps mit einer dezimalen Genauigkeit von beispielsweise 15 signifikanten Nachkommastellen abgekürzt als $d_{A,B}^{RK10,15}(t)$ geschrieben werden.

8.1.4.1 Resultate bei einer Simulationsdauer von einem Tag (16 Umläufe)

Wird der Simulationszeitraum auf einen Tag begrenzt, so ergeben sich die in Abbildung 8.3 dargestellten Abstandsdifferenzen. Hierbei wurde für jedes der verwendeten Integrationsverfahren zunächst eine genauere Lösung errechnet, bei der 32 signifikante Nachkommastellen berücksichtigt wurden. Im nächsten Schritt wurde dann eine ungenauere Lösung mit 16 signifikanten Nachkommastellen Genauigkeit zur Berechnung verwendet. Somit liegen nach den ersten beiden Schritten die Abstände $d_{A,B}^{I1,32}(t)$ und $d_{A,B}^{I2,16}(t)$ vor. Als Integrationsverfahren kamen SG, BS, RK10 und RK4 zum Einsatz.

[SG] Shampine-Gordon
[BS] Burlisch-Stoer
[RK10] Runge-Kutta 10.Ordnung
[RK4] Runge-Kutta 4.Ordnung

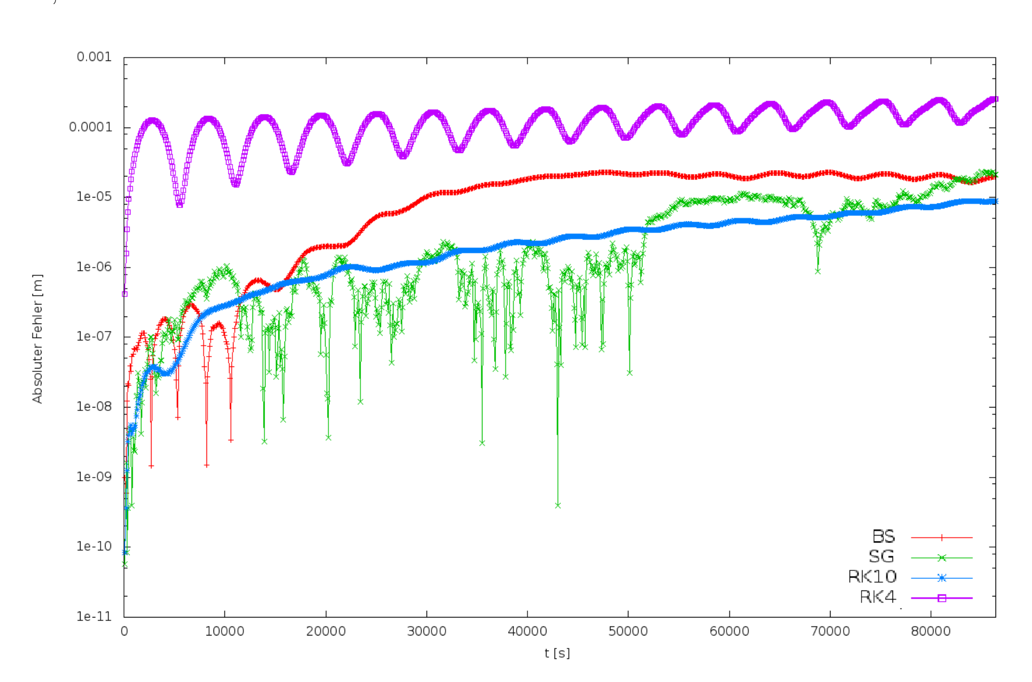


Abbildung 8.3: Abstandsdifferenzen verschiedener Lösungen über einen Simulationszeitraum von einem Tag (16 Umläufe)

Nach der Simulation stehen Datensätze, aus denen Abstandsdifferenzen gebildet werden können, zur Verfügung:

$$\begin{array}{ll} d_{A,B}^{SG,64}(t) & d_{A,B}^{SG,16}(t) \\ d_{A,B}^{BS,64}(t) & d_{A,B}^{BS,16}(t) \\ d_{A,B}^{RK10,64}(t) & d_{A,B}^{RK10,16}(t) \\ d_{A,B}^{RK4,64}(t) & d_{A,B}^{RK4,16}(t) \end{array}$$

Die Abstandsdifferenzen aus Abb. 8.3 wurden mit der hochgenauen Lösung $d_{A,B}^{SG,64}$

verglichen. Dadurch ergeben sich die Abstandsdifferenzen:

$$d_{A,B}^{SG,64,SG,16}(t)$$
$$d_{A,B}^{SG,64,BS,16}(t)$$
$$d_{A,B}^{SG,64,RK10,16}(t)$$
$$d_{A,B}^{SG,64,RK4,16}(t)$$

Bei genauer Betrachtung der Abstandsdifferenzen fällt auf, dass das RK4-Verfahren trotz höherer dezimaler Genauigkeit, verglichen mit den anderen Verfahren, größere Fehler aufweist. Der Grund hierfür liegt in der vergleichsweise geringen Ordnung des Verfahrens. Somit kann das RK10-Verfahren, aufgrund seiner größeren Ordnung, genauere Ergebnisse erzielen. Weiterhin fällt auf, dass die Verfahren mit adaptiver Schrittweitensteuerung stärkeren Schwankungen unterliegen. Im Allgemeinen erzielen die Verfahren mit adaptiver Schrittweitensteuerung ein besseres Fehlerverhalten, was hier ebenfalls abzulesen ist.

8.1.4.2 Resultate bei einer Simulationsdauer von sieben Tagen (122 Umläufe)

In diesem Abschnitt wird der Integrationszeitraum auf sieben Tage erweitert, um den Verlauf der Abstandsdifferenzen über einen längeren Zeitraum zu beurteilen. Die resultierenden Abstandsdifferenzen sind in Abb. 8.4 dargestellt. Hier zeigt sich bei den

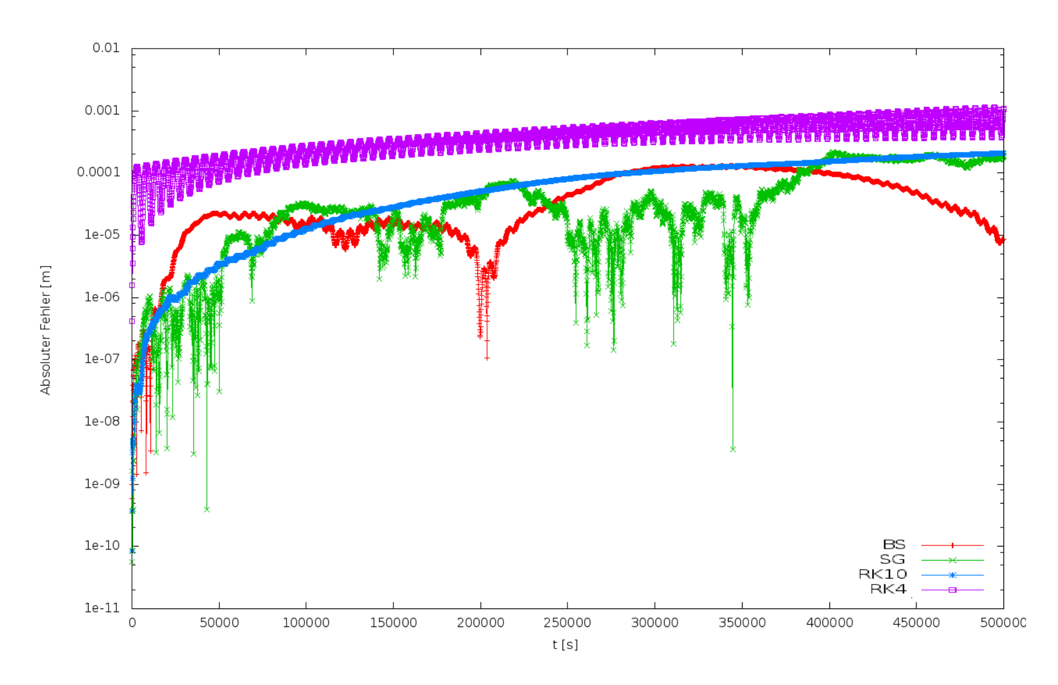


Abbildung 8.4: Abstandsdifferenzen verschiedener Lösungen in einem Simulationszeitraum von sieben Tagen (122 Umläufe)

Verfahren mit fester Ordnung ohne adaptive Schrittweitensteuerung ein ähnliches Verhalten wie im vorausgegangen untersuchten Zeitabschnitt. Die Runge-Kutta-Verfahren können aufgrund ihrer festgelegten Ordnung, auch bei Vergrößerung der signifikanten

Nachkommastellen, kein genaueres Ergebnis erzielen. Wird eine Integration mit dem RK4-Verfahren unter Verwendung herkömmlicher Zahlenlänge durchgeführt, so liegt der absolute Fehler der Abstandsdifferenz nach sieben Tagen bei mehr als einem Millimeter. Auch die Verfahren mit adaptiver Schrittweitensteuerung (SG und BS) sind bei diesem Simulationszeitraum deutlich schlechter als 1 Mikrometer, wobei das BS-Verfahren noch den kleinsten absoluten Fehler aufweist.

8.1.5 Fazit

Im Hinblick auf geplante GRACE-Nachfolgemissionen[4], deren Genauigkeiten aufgrund technischer Verbesserungen und einem zusätzlich an Bord befindlichen Laserentfernungsmesssystem[5] im Nanometer-Bereich liegen sollen, müssen die numerischen Methoden weiter verbessert werden.

8.2 Berechnung einer Ephemeride der Mondbewegung

8.2.1 Historie des Programmpakets *LUNAR*

Zur Auswertung und Validierung von Laserentfernungsmessungen zum Mond (LLR) wurde an der Forschungseinrichtung Satellitengeodäsie der Technischen Universität München unter Leitung von Prof. Dr. M. Schneider das Programmpaket *LUNAR* entwickelt. Dieses wurde als Teil der Dissertation von Herrn B. Reichhoff[6] weiter verbessert und in der Programmiersprache C++ neu verfasst. Insgesamt besteht dieses Programmpaket aus den drei Modulen: *VARIAT*, *NORMAL* und *EPHEMER*, wobei letzteres im Rahmen der vorliegenden Arbeit verwendet wurde. Dessen Funktionsweise wird im Folgenden kurz beschrieben.

[LLR] Lunar Laser Ranging

- *EPHEMER*
  Mit diesem Modul kann u.a. der Abstand zwischen Erde und Mond simuliert werden. Der theoretische Hintergrund des hierbei implementierten Modells ist in [Gleixner86], [Bauer89], [Müller91] und [Reichhoff99] nachzulesen. Aufgrund der gravitativen Wechselwirkung zwischen Erde, Mond und der Sonne mit den restlichen Planeten im Sonnensystem müssen deren Bewegungen bei jedem Berechnungsschritt berücksichtigt werden.

In Abb. 8.5 ist die Erde - Mond - Distanz-Berechnung schematisch illustriert. Dabei ist R_M der geozentrische Ortsvektor des Mondes bezüglich der Ekliptik. Das Programmpaket wurde im Rahmen dieser Arbeit auf ein aktuelles Linux-System[7] portiert, da es aufgrund der sich ständig ändernden Compilerversionen und den damit verbundenen Bibliotheksabhängigkeiten nicht mehr übersetzbar war. Hierzu war es außerdem nötig, die LiDIA-Bibliothek der Version 1.3 auf das aktuelle Betriebssystem zu portieren.

[4] siehe mehr dazu unter [Watkins10]
[5] siehe hierzu http://rses.anu.edu.au/geodynamics/GRACE_Follow_On/tech.php Download am 25.11.2011
[6] siehe mehr dazu in der Dissertation von B. Reichhoff [Reichhoff99]
[7] Ubuntu Linux 10.10 (32-Bit), gcc/g++-4.2

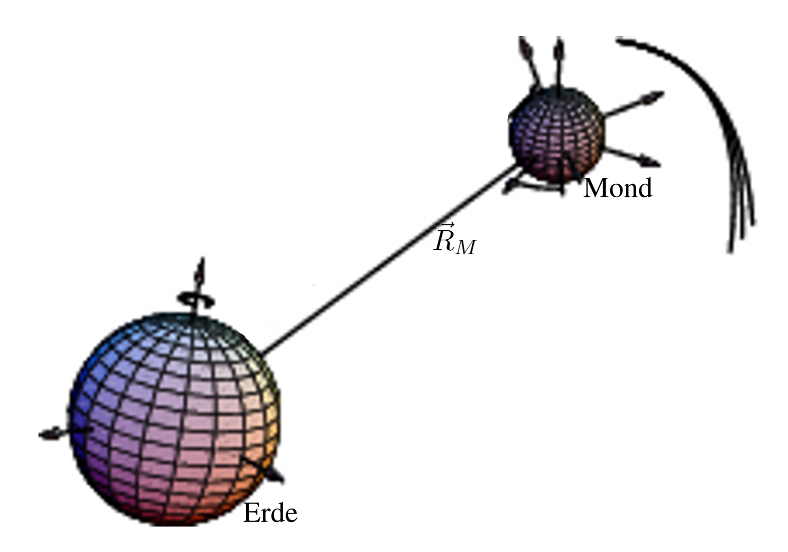


Abbildung 8.5: Illustration zum Abstand Erde-Mond (bearbeitet und entnommen aus der Dissertation von B. Reichhoff)

8.2.2 Berechnung einer Ephemeride durch Integration im herkömmlichen Modus

In diesem Abschnitt soll *EPHEMER* verwendet werden, um über einen Zeitraum von 32 Jahren eine Ephemeride des Mondes zu berechnen. Dabei werden alle relevanten gravitativen Einflussgrößen mit Post-Newtonscher Näherung im Sonnensystem berücksichtigt. Als Integrationsverfahren wird das ABM-Verfahren verwendet. Insgesamt werden zwei verschiedene Lösungen berechnet. Die erste Lösung wird mit Hilfe des standardmäßig verfügbaren *long double*-Datentyps bestimmt, was folglich eine absolute Fehlertoleranz von 1.0e-18 fordert. Die zweite Lösung wird mit Hilfe des *bigfloat*-Datentyps der LiDIA-Bibliothek errechnet. Hierbei wird die Anzahl signifikanter Nachkommastellen auf 80 und die geforderte absolute Fehlertoleranz auf 1.0e-38 festgelegt.

[ABM] Adams Bashforth Moulton

Das Ergebnis des Vergleichs der beiden Lösungen ist in Abbildung 8.6 dargestellt. In den ersten drei Graphen ist der absolute Fehler der Koordinatenunterschiede aufgetragen. Dabei fällt auf, dass der größte Fehlerbeitrag in der x und y - Koordinate liegt und sich die Fehlerentwicklung der drei Koordinatenachsen, trotz unterschiedlicher Größenordnung in der z-Koordinate, gleich verhält. Dies lässt auf einen Fehler in *alongtrack*-Richtung schließen[8].

Außerdem ist im vierten Plot (Abb. 8.6, unten) der absolute Fehler für den Erde-Mond-Abstand aufgetragen. Dort fällt auf, dass sich nach ca. 14 und 24 Jahren der absolute Fehler auf etwa demselben Niveau (0,25 Meter) befindet, daraufhin absinkt und anschließend steil auf 0.5 Meter ansteigt. Dies bedeutet: Wird die Simulation mit herkömmlicher Genauigkeit *long double* durchgeführt, dann besitzt das Ergebnis eine Ungenauigkeit von 0.55 Meter[9]. Soll über diesen Zeitraum hinweg integriert werden, so ist die Berechnung mit mehrfacher Genauigkeit durchzuführen.

[8] Siehe dazu die Phasenverschiebung der x und y - Komponente in Abb. F.13 auf Seite 225

[9] siehe dazu auch [Ettl10]

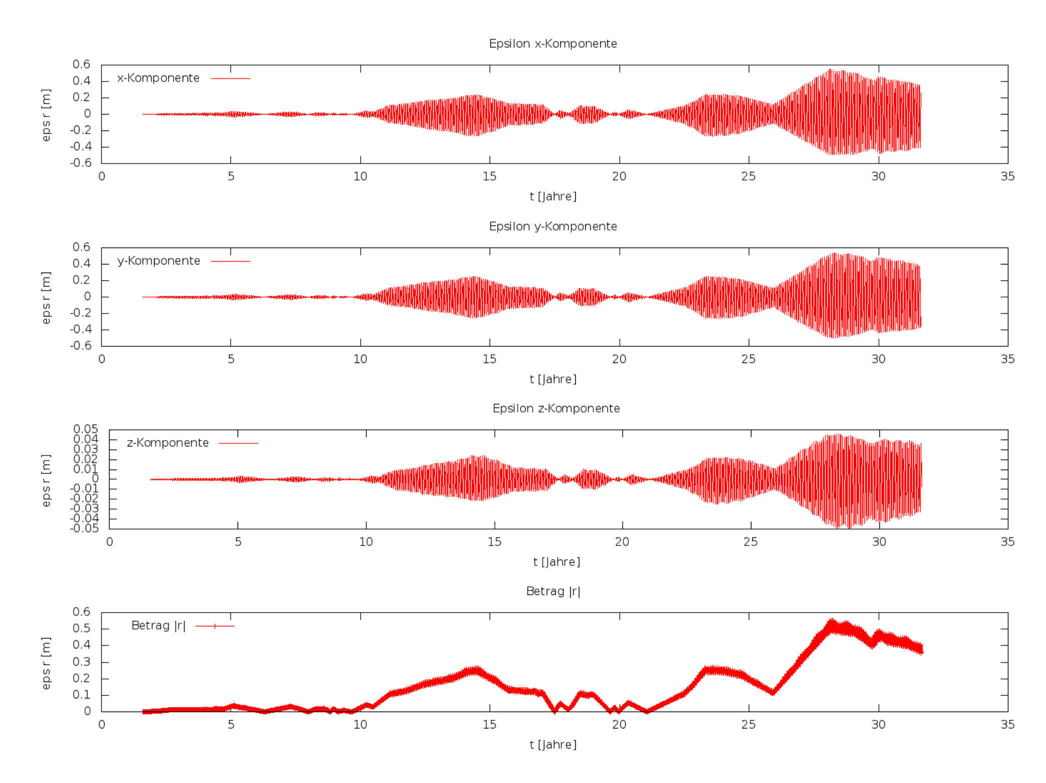


Abbildung 8.6: Vergleich zweier Lösungen des Erde-Mond Abstands

8.2.3 Erweiterung von *EPHEMER*

Bei der Verarbeitung der Messdaten muss die Bewegung des in das Sonnensystem eingebettete Erde-Mond-Systems in nachnewtonscher Näherung gelöst werden. Die iterative Bestimmung von Modellparametern erfolgt nach dem Verfahren der differentiellen Korrektur von apriori-Werten. Dabei fallen bei jedem Schritt eine Ephemeridenrechnung des gesamten Sonnensystems, eine Berechnung der partiellen Ableitungen der testbaren Funktionale nach den Modellparametern sowie eine Ausgleichsrechnung der Beobachtungsgleichungen an. Die Iteration konvergiert nur, wenn die partiellen Ableitungen, die von der Genauigkeit der Ephemeridenrechnung abhängen, hinreichend genau gerechnet werden. Hierzu wird im Folgenden das Programm *EPHEMPER* erweitert, um zu überprüfen, welche Genauigkeit (exakte Anfangsbedingungen vorausgesetzt) bei Verwendung des double-Datentyps erreichbar ist.
Die hochgenaue Berechnung einer Ephemeride über einen längeren Zeitraum von beispielsweise 30 Jahren ist auch bei heutigen[10] Rechnersystemen[11] noch sehr zeitaufwendig. Beispielsweise benötigt die Berechnung einer Ephemeride mit dem ABM-Integrator (Absolute Fehlertoleranz bei 1.0e-22) unter Verwendung des bigfloat-Datentyps bei eingestellten 50 signifikanten Nachkommastellen auf TP1 etwa 12 Stunden.
Die neu hinzugefügte Funktionalität von *EPHEMER* wird als Zweischritt-Modus bezeichnet. Die Arbeitsweise ist außerdem in Abb. 8.7 illustriert.
Im ersten Arbeitsschritt wird eine Ephemeride mit hoher Anzahl signifikanter Nach-

[ABM] Adams Bashforth Moulton

[TP1] Testplattform 1

[10] Stand November 2011
[11] siehe mehr dazu im Abschnitt A.1.

kommastellen berechnet. Hier wird analog zum klassischen Betriebsmodus der gesamte Zeitraum integriert, wobei in Abständen von einem Jahr Integrationszeit die jeweiligen Positionsvektoren der Planeten im Sonnensystem in einer entsprechenden Datei abgelegt werden.

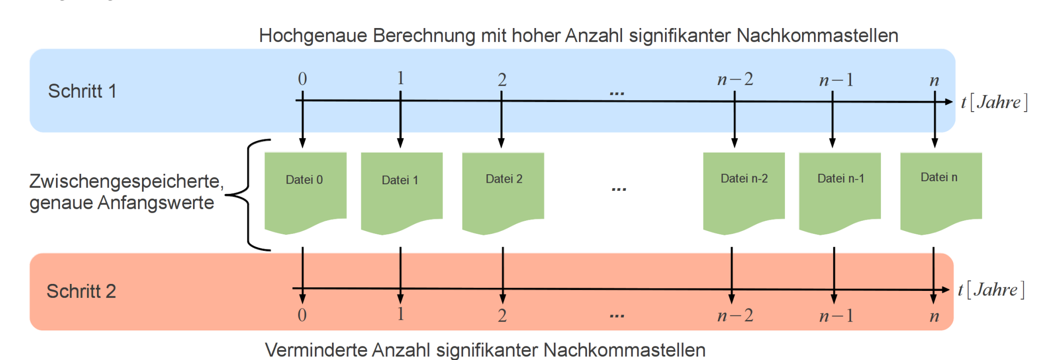


Abbildung 8.7: Funktionsweise der EPHEMER Erweiterung

Im zweiten Schritt können die zwischengespeicherten Positionsvektoren aus dem ersten Schritt verwendet werden, um bei Rechnung mit herkömmlicher Genauigkeit (*long double*-Datentyp) nach jeweils einem Jahr Integrationszeit mit genaueren Anfangswerten zu beginnen. Dadurch wird erstens kostbare Rechenzeit eingespart und zweitens ein genaueres Ergebnis erzielt als mit herkömmlicher Genauigkeit. In Abb.

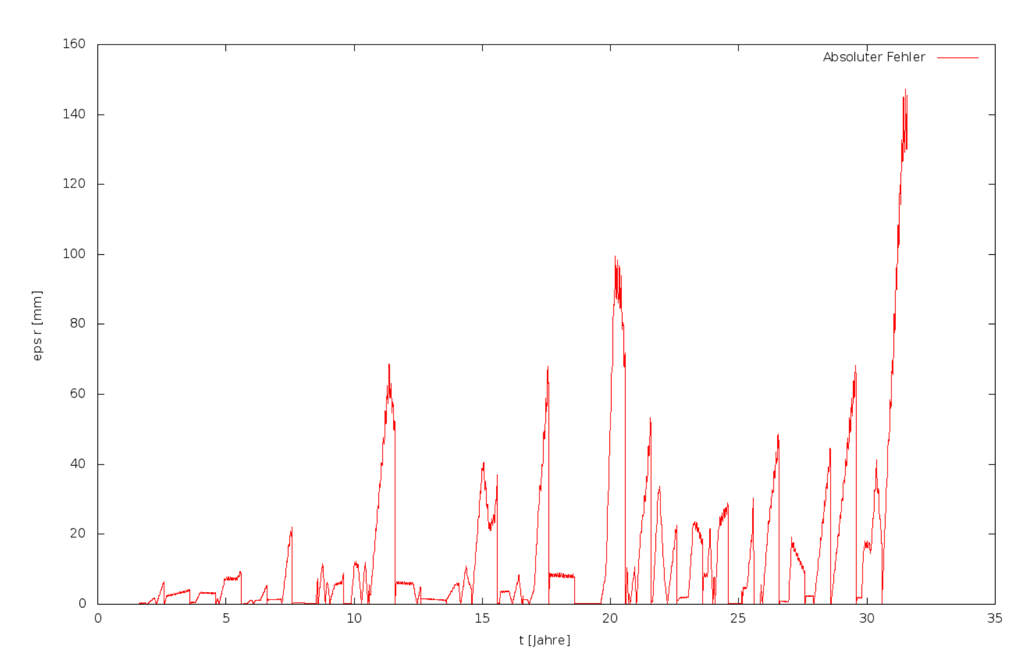


Abbildung 8.8: Qualität der Lösung bei der Berechnung im Zweischritt-Modus (Jahresschritte)

8.8 ist das Ergebnis aus dem Vergleich der hochgenauen Lösung einer Ephemeride mit dem Ergebnis aus dem Zweischrittmodus dargestellt. Hierbei kam im zweiten Schritt der *long double*-Datentyp und der ABM-Integrator mit einer geforderten absoluten Fehlertoleranz von 1.0e-19 zum Einsatz. Die hierfür benötigten Ergebnisse wurden

im ersten Schritt, bei sonst identischer Konfiguration, mit dem *bigfloat*-Datentyp und einer geforderten absoluten Fehlertoleranz von 1.0e-22 erstellt. Im gesamten Integrationszeitraum von 32 Jahren liegt die erzielte Genauigkeit bei ca. 14 cm. Auffällig sind dennoch die Anstiege bei 12, 15, 20 und 30 Jahren. Dieser erste Versuch zeigt, dass die hierdurch erzielte Genauigkeit noch nicht ausreicht, um mit den derzeit erreichbaren Genauigkeiten der Erde-Mond Messungen von ca. 1 cm schritthalten zu können. Aus diesem Grund wird eine weitere Modifikation vorgenommen. Anstatt in Abständen von einem Jahr hochgenaue Positionsvektoren zu laden, wird die Integration mit hochgenauen Anfangswerten in Monatsabständen verwendet. Der Vergleich der hochgenauen Lösung einer Ephemeride mit dem Ergebnis aus dem Zweischrittmodus (Monatsabständen) ist in Abb. 8.9 dargestellt.

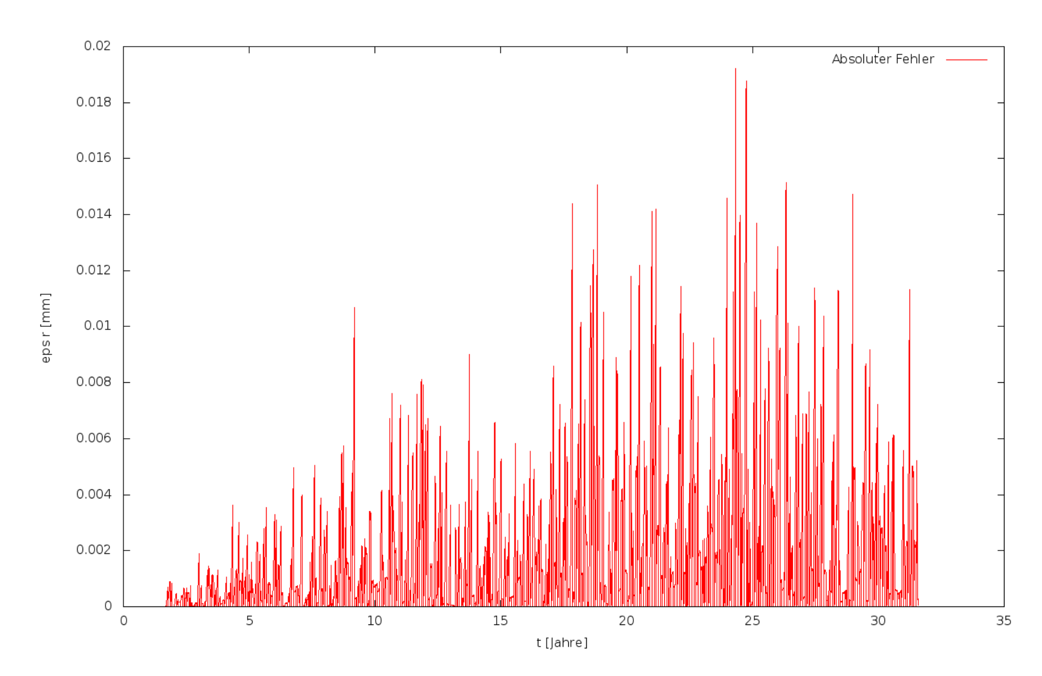


Abbildung 8.9: Qualität der Lösung bei der Berechnung im Zweischritt-Modus (Monatsschritte)

Daraus wird ersichtlich, dass im gesamten Zeitraum der Integration die erzielte Genauigkeit unter 0.02 mm liegt. Somit kann diese Variante der Ephemeridenberechnung als guter Kompromiss zwischen erzielter Genauigkeit und benötigtem Rechenaufwand betrachtet werden. Wie diese Beispiele gezeigt haben, ist jedoch darauf zu achten, in welchen zeitlichen Abständen die Zwischenschritte verwendet werden.

8.2.4 Einfluss des impliziten Rundungsfehlers auf die Berechnung des Erde-Mond Abstands

In einem weiteren Experiment wurde das EPHEMER-Programmpaket verwendet, um den Einfluss impliziten Rundens auf die Berechnung des Erde-Mond Abstands zu untersuchen. Eine Zahl wird implizit gerundet, falls ihre tatsächliche Darstellung nicht auf die Mantissenbits der Gleitkommazahlendarstellung abgebildet werden kann. Diese spezielle Form der Rundung tritt ausschließlich beim Basiswechsel zwischen Dezi-

mal in Binärsystem auf. Ein Beispiel hierfür wurde bereits in Abschnitt 3.1.3 auf Seite 69 gegeben.
Wie in den vorangegangen Experimenten wird die jeweilige Berechnung des Erde-Mond Abstands auf zwei unterschiedliche Arten durchgeführt. Bei der ersten Variante der Berechnung werden alle benötigten Anfangswerte, Koeffizienten und Konstanten mit Hilfe von ASCII-Zeichenketten initialisiert. Damit ist es möglich, das implizite Runden im Zusammenspiel mit dem *bigfloat*-Datentyp und einer entsprechenden Anzahl signifikanter Nachkommastellen zu verhindern. Bei der zweiten Variante wird implizites Runden explizit erlaubt und es werden alle notwendigen Anfangswerte in üblicher Weise initialisiert. Außerdem wurde in allen durchgeführten Berechnungen der ABM-Integrator mit einer absoluten Fehlertoleranz von 1.0e-22 eingesetzt. Als Multiprecision-Datentyp wurde der *bigfloat*-Datentyp der LiDIA-Bibliothek verwendet und auf 80 signifikante Nachkommastellen eingestellt.

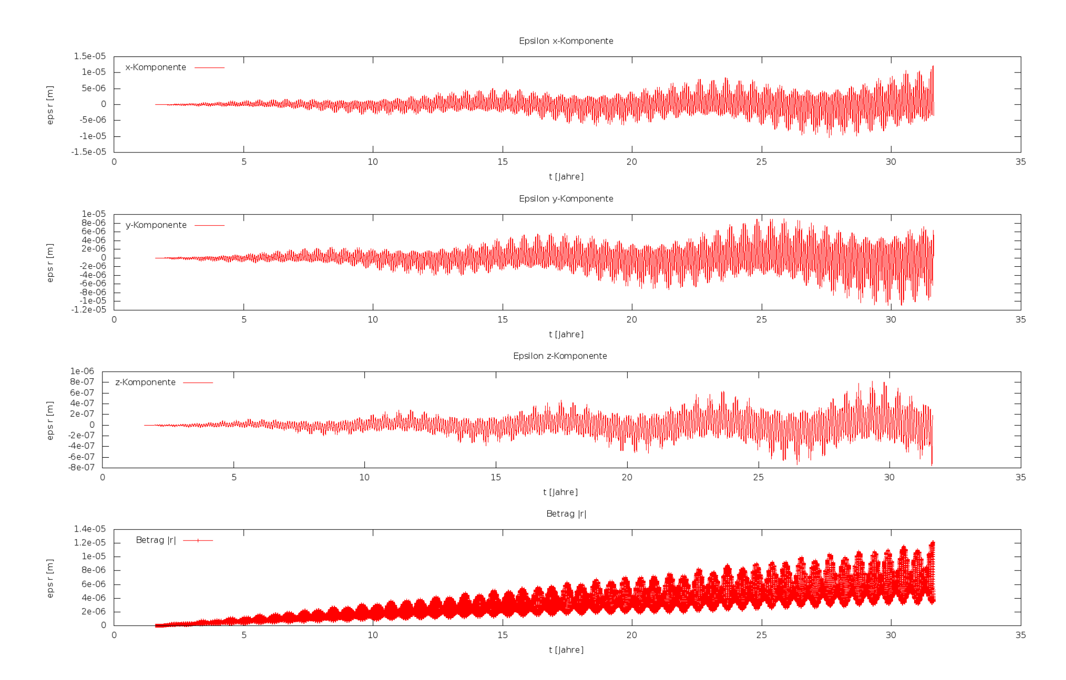


Abbildung 8.10: Einfluss des impliziten Rundungsfehlers auf die Berechnung des Erde-Mond Abstands

In Abb. 8.10 sind die Ergebnisse des Vergleichs der beiden Berechnungen illustriert. Die ersten drei Graphen stellen den absoluten Fehler in den kartesischen Koordinaten dar. Hier ist analog zur vorausgegangenen Berechnung (siehe Seite 8.6) der Fehlerbeitrag der x und y - Komponente größer als die der z-Komponente. In der vierten Graphik (Abb. 8.10) ist der absolute Fehler des geozentrischen Abstands aufgetragen. Dieser liegt maximal bei 1.4e-5 Meter (0.014 mm) und ist somit kein potentieller Einflussfaktor für verfälschte Ergebnisse bei Berechnungen dieser Art. Außerdem soll noch bemerkt werden, dass der absolute Fehler aus der Berechnung der Monatslösung (siehe Abb. 8.9) die selbe Größenordnung aufweist.

Kapitel 9

Ausblick

In den vorausgegangen Kapiteln wurde das Konzept einer *Toolbox* zur numerischen Lösung gewöhnlicher Differentialgleichungen vorgestellt. Ihre Einsatzbereitschaft wurde insbesondere anhand der Abstandssimulation der beiden GRACE-Satelliten und anhand der Berechnung der Erde-Mond-Distanz demonstriert. Daraus hat sich gezeigt, dass bei der Rechnung mit 16 signifikanten Nachkommastellen die Ergebnisse nicht immer den geforderten Genauigkeiten genügen. Dieser Nachteil kann durch den Einsatz von *multiprecision*-Bibliotheken beseitigt werden. Die *Toolbox* besteht sowohl aus Ein- und Mehrschrittverfahren als auch aus Potenzreihenintegrationsverfahren, welche aufgrund des erarbeiteten objektorientierten Designs mit verschiedenen *multiprecision*-Bibliotheken verwendet werden können. Insbesondere der zusätzliche Freiheitsgrad der frei wählbaren Anzahl signifikanter Nachkommastellen bei der numerischen Integration hat sich als wertvolles Hilfsmittel erwiesen, Ergebnisse zu validieren und höhere Genauigkeiten zu erreichen. Außerdem können damit bestehende Programmsysteme hinsichtlich ihrer numerischen Stabilität überprüft werden. Der zeitliche Mehraufwand bei der Berechnung mit höherer Anzahl signifikanter Nachkommastellen kann durch die vorgestellten Methoden zur Laufzeitverbesserung in vielen Fällen abgeschwächt werden.
Im Folgenden sollen weiterführende Gedanken zur untersuchten Thematik angesprochen werden, die in zukünftigen Arbeiten zu diesen oder verwandten Themenstellungen angegangen werden könnten.

9.1 Internationaler Standard für numerische Verfahren

Es gibt unzählige Veröffentlichungen, in denen numerische Integrationsverfahren als Hilfsmittel eingesetzt wurden, um gewöhnliche Differentialgleichungen aller Art zu lösen. Die hierfür verwendeten Verfahren entstammen dabei aus unterschiedlichen Quellen, wurden mit verschiedenen Programmiersprachen umgesetzt und oft auf nicht näher spezifizierten Computerplattformen berechnet. Folglich besteht die Notwendigkeit, einen internationalen Standard für numerische Verfahren, ähnlich dem IEEE Standard für Arithmetik von Gleitkommazahlen, auszuarbeiten. In diesem sollten demnach die Standardverfahren als Pseudocode definiert sein. Daraus könnten dann für die jeweiligen Programmiersprachen Bibliotheken entwickelt werden. Damit wären die Ergebnisse, welche auf unterschiedlichen Rechnersystemen und Programmiersprachen erstellt wurden, besser vergleichbar.

9.2 Schnelles Rechnen

[GPU] Graphics Processing Unit
[FPGA] field programmable gate array

Höhere Rechengeschwindigkeiten spielen insbesondere bei der Simulation mit hoher Anzahl signifikanter Nachkommastellen eine entscheidende Rolle. Aus diesem Grund erscheinen die Berechnungsmöglichkeiten des GPU-Computings, basierend auf der OpenACC-Schnittstelle, zukunftsweisend. Ferner könnten Programme auf FPGA-Bausteinen abgebildet werden und dadurch einen erheblichen Geschwindigkeitszuwachs erfahren.

9.3 Untersuchung zu *multiprecision*-Bibliotheken

In dieser Arbeit wurden Untersuchungen zur Eignung einiger frei verfügbarer *multiprecision*-Bibliotheken angestellt. Diese Vergleiche könnten auf eine größere Menge Bibliotheken und Programmiersprachen ausgeweitet werden. Dadurch könnten Informationen zur momentanen Leistungsfähigkeit heutiger Bibliotheken gesammelt werden, die u.U. als Entscheidungshilfe für spätere Projektvorhaben dienen könnten.
Ein weiteres Entwicklungsfeld stellt die automatische Validierung errechneter Ergebnisse unterschiedlicher *multiprecision*-Bibliotheken dar. Hierzu wäre es erforderlich, für die jeweiligen Bibliotheken, Testprogramme zu entwickeln, welche auf verschiedenen Plattformen mit Hilfe von *unit*-Tests die korrekte Arbeitsweise der Bibliotheken nachweisen. Somit könnten versteckte Ungenauigkeiten schon im Vorfeld detektiert werden, was die Vertrauenswürdigkeit derartig bestimmter Ergebnisse nachhaltig fördern würde. Mögliche Ansatzpunkte wären die Weiterentwicklung des Programms *Arithmos*[1], mit dem mathematische Berechnungen mit Hilfe hochgenauer Rechnung validiert werden können.
Eine zusätzliche Möglichkeit könnte die Erweiterung einer Compilersuite um einen *multiprecision*-Datentyp sein. Dessen jeweilige Genauigkeit könnte beispielsweise beim Übersetzungsvorgang angegeben werden. Dies würde die Entwicklung numerischer Berechnung mit C/C++ deutlich vereinfachen.

9.4 Erweiterung der Integratoren

Im Rahmen dieser Arbeit wurden eine ganze Reihe verschiedener Integrationsverfahren implementiert. Es gibt dennoch einige Verfahren, die noch nicht im Repertoire enthalten sind. Dazu gehören beispielsweise das Runge-Kutta-Shanks-Verfahren[2] oder der Verlet-Integrator. Darüber hinaus fehlen noch Verfahren, vom Gauß-Jackson Verfahren abgesehen, zur Lösung von Differentialgleichungen 2.Ordnung. Außerdem wäre es überaus hilfreich, eine adaptive Schrittweitensteuerung für den Potenzreihenintegrator zu implementieren.

[1] siehe mehr dazu auf der Webseite des Projekts: http://cant.ua.ac.be/old/arithmos/, Download am 26.11.2011
[2] siehe mehr dazu unter [Schneider99], Seite 1047

9.5 Erweiterung der Toolbox

Die Toolbox könnte in einer weiteren Ausbaustufe um Verfahren zur Lösung von Randwertproblemen aufgestockt werden. Damit könnte es, aufgrund der Möglichkeit mit höherer Anzahl signifikanter Nachkommastellen zu rechnen, ein nützliches Werkzeug bei Bahn - und Parameterbestimmung von Ephemeriden werden.
Zusätzlich wäre es hilfreich, das Programm EPHEMER und die darin enthaltenen Bewegungsgleichungen auf C++-Templates umzustellen. Ferner sollten die verfügbaren Bewegungsgleichungen um zusätzliche Kraftanteile, wie beispielsweise den Einfluss des Strahlungsdrucks der Sonne, erweitert werden.

9.6 Graphische Benutzeroberfläche

Schließlich wäre es lohnenswert, das entwickelte Programmsystem mit einer graphischen Benutzeroberfläche auszustatten. Dadurch würde die Bedienung erheblich erleichtert und wäre folglich einem breiteren Benutzerkreis zugänglich. Als potentielle Bibliotheken zur Realisierung einer graphischen Benutzeroberfläche werden *wxWidgets*[3] oder *Qt*[4] angesehen.

[3] siehe mehr dazu auf der Webseite des Projekts: http://www.wxwidgets.org, Download am 26.11.2011
[4] siehe mehr dazu auf der Webseite des Projekts: http://qt.nokia.com/, Download am 26.11.2011

Kapitel 10

Danksagung

Mein Dank gebührt vor allem Herrn Prof. Dr. rer. nat. M. Schneider, der diese Dissertation angeregt hat und mir stets geduldig mit Ratschlägen und nützlichen Hinweisen zur einschlägigen Fachliteratur zur Seite stand. Er hatte immer ein offenes Ohr für meine Fragen.
Ich bedanke mich außerdem bei Herrn Univ.-Prof. Dr. phil. nat. Urs Hugentobler, der sich immer Zeit für meine Fragen nahm und mich während der Erstellung dieser Arbeit stets unterstützte. Außerdem möchte ich mich bei Herrn Univ.-Prof. Dr. Hans-Joachim Bungartz für die wertvollen Impulse zur Arbeit bedanken.

Anhang A

Anhang

A.1 Beschreibung der Testplattformen

Nachfolgend sind die Leistungsmerkmale der eingesetzten Testplattformen aufgelistet. Durch Angabe dieser Merkmale erhalten die Geschwindigkeitsmessungen ihre Aussagekraft. Ferner ist es dadurch jederzeit möglich, die Ergebnisse einer nachträglichen Verifikation zu unterziehen.

A.1.1 Testplattform 1

Betriebssystem	Ubuntu Linux 7.10
Kernelversion	2.6.22-14-generic x86-64
Prozessor	Intel Core 2 Duo T7300 [Family 6 Model 15 Stepping 10]
Arbeitsspeicher	2.0 GB
Compilerversion	gcc-4.2.1 / g++-4.2.1
Compilerflags	-m64

Tabelle A.1: Eckdaten der Testplattform 1

A.1.2 Testplattform 2

Diese Testplattform besitzt eine CPU mit HT-Technologie. Die HT-Technologie stellt eine Implementierung von hardwareseitigem Multithreading dar. Mit anderen Worten, es wird ein zweiter Rechenkern simuliert. Dadurch kann diese Plattform auch als Testumgebung zur Parallelverarbeitung eingesetzt werden. Dieses Feature wurde bei allen Berechnungen eingeschaltet.

[HT] Hyperthreading

Betriebssystem	Debian Edge
Kernelversion	2.6.17.1
Prozessor	Pentium 4 3.0GHz (HT)
Arbeitsspeicher	1.0 GB
Compilerversion	gcc-4.2.1 / g++-4.2.1
Compilerflags	-m32

Tabelle A.2: Eckdaten der Testplattform 2

Anhang B

Formale Darstellung der Rücksubstitution

Eine Differentialgleichung kann mit Hilfe eine Taylorreihenentwicklung gelöst werden. Dazu ist es erforderlich, analytische Ableitungen höherer Ordnung zu bestimmen. Betrachtet man die analytisch bestimmten Ableitungsterme genauer, so stellt man fest, dass diese nicht nur von Ort ($\frac{d^0x}{dt^0}$) und Geschwindigkeit ($\frac{d^1x}{dt^1}$), sondern auch von den Ableitungstermen niederer Ordnung implizit abhängig sind. Durch diese Abhängigkeit müssen die vorangegangenen Ableitungen bekannt sein, um die nächsthöhere zu bilden. Folglich ist, zumindest in dieser Darstellung, die Berechnung einzelner Ableitungsterme nicht zur parallelen Abarbeitung geeignet. Im Folgenden soll aufgezeigt werden, wie durch sukzessives Rückwärtseinsetzen (Rücksubstitution) Ableitungsterme höherer Ordnung berechnet werden können, die zur Parallelverarbeitung geeignet sind. Ferner wird ein Bildungsgesetz aufgezeigt, welches die Berechnung eines bestimmten Reihenkoeffizienten i ermöglicht, ohne die vorangegangenen Ableitungsterme niedrigerer Ordnung zu kennen.

B.1 Vollständiges Differential

Wenn $u(x, y)$ eine differenzierbare Funktion ist und von den unabhängigen Variablen x und y abhängig ist, so wird das vollständige Differential ($D^{(1)}$) erster Ordnung durch partielle Ableitungen wie folgt ausgedrückt:

$$D^{(1)} := du = \frac{\partial u}{\partial x}dx + \frac{\partial u}{\partial y}dy \tag{B.1.1}$$

Höhere partielle Ableitungen der Ordnung n werden rekursiv definiert durch:

$$D^{(n)} = \frac{\partial D^{(n-1)}}{\partial x}dx + \frac{\partial D^{(n-1)}}{\partial y}dy \tag{B.1.2}$$

Voraussetzung hierzu ist:

Werden höhere partielle Ableitungen gebildet, so kann erst nach einer unabhängigen Variablen abgleitet werden und schließlich nach einer anderen. Diese Form wird als gemischt partielle Ableitung bezeichnet. Eine partielle Ableitung nach x und dann nach y wird beschrieben durch:

$$\frac{\partial^2 u}{\partial x \partial y} = \frac{\partial^2 u}{\partial y \partial x}$$

Der Wert einer gemischten Ableitung ist für gegebene Werte von x und y unabhängig von der Reihenfolge der Ableitungsbildung, wenn die gemischte Ableitung in den betrachteten Punkt stetig ist[1].

B.1.1 Rücksubstitution

Zur Lösung einer Bewegungsgleichung (Differentialgleichung) 2.Ordnung werden folgende Anfangswerte benötigt:

- Ort u
- Geschwindigkeit $\dot{u}$

Außerdem wird die Differentialgleichung $\ddot{u}(u, \dot{u})$ in den weiteren Schritten benötigt. Sind diese Angaben vorhanden, so können zunächst höhere Ableitungen gebildet werden:

$$\frac{d^1\ddot{u}}{dt^1} = \frac{d^1\ddot{u}(u,\dot{u})}{dt^1}$$
$$\frac{d^2\ddot{u}}{dt^2} = \frac{d^2\ddot{u}(u,\dot{u})}{dt^2}$$
$$...$$
$$\frac{d^n\ddot{u}}{dt^n} = \frac{d^n\ddot{u}(u,\dot{u})}{dt^n}$$

Aus dieser Darstellung wird die Abhängigkeit zwischen den unterschiedlichen Ordnungen ersichtlich, d.h. um die Ableitung der Ordnung n zu bilden, werden die vorangegangen Ableitungen der Ordnungen $< n$ benötigt. Außerdem erhält man Ableitungen, die implizit von niedrigeren Ordnungen abhängen.
Mit Hilfe des vollständigen Differentials lässt sich folgende Darstellung gewinnen:

$$\frac{d^1\ddot{u}}{dt^1} = \frac{\partial\ddot{u}}{\partial u}\dot{u} + \frac{\partial\ddot{u}}{\partial\dot{u}}\ddot{u}$$
$$\frac{d^2\ddot{u}}{dt^2} = \frac{\partial^2}{\partial u^2}(\frac{d^1\ddot{u}}{dt^1})\dot{u} + \frac{\partial^2}{\partial\dot{u}^2}\ddot{u}$$
$$...$$
$$\frac{d^n\ddot{u}}{dt^n} = \frac{\partial^n}{\partial u^n}(\frac{d^{n-1}\ddot{u}}{dt^{n-1}})\dot{u} + \frac{\partial^n}{\partial\dot{u}^n}(\frac{d^{n-1}\ddot{u}}{dt^{n-1}})\ddot{u}$$

Soll beispielsweise der 5. Ableitungsterm berechnet werden, dann ergeben sich folgende Zwischenergebnisse:

$$\frac{d^5\ddot{u}}{dt^5} = \frac{\partial}{\partial u}(\frac{d^4\ddot{u}}{dt^4})\dot{u} + \frac{\partial}{\partial\dot{u}}(\frac{d^4\ddot{u}}{dt^4})\ddot{u}$$

Wird die partielle Ableitung der 4.Ordnung eingesetzt:

$$\frac{d^5\ddot{u}}{dt^5} = \frac{\partial}{\partial u}[\frac{\partial}{\partial u}(\frac{d^3\ddot{u}}{dt^3})\dot{u} + \frac{\partial}{\partial\dot{u}}(\frac{d^3\ddot{u}}{dt^3})\ddot{u}]\dot{u} + \frac{\partial}{\partial\dot{u}}[\frac{\partial}{\partial u}(\frac{d^3\ddot{u}}{dt^3})\dot{u} + \frac{\partial}{\partial\dot{u}}(\frac{d^3\ddot{u}}{dt^3})\ddot{u}]\ddot{u}$$

[1] siehe dazu [Bronstein08]

Nach Einsetzen der 3.Ableitung:

$$\begin{aligned}\frac{d^5\ddot{u}}{dt^5} = {} & \frac{\partial}{\partial u}[\frac{\partial}{\partial u}(\frac{\partial}{\partial u}(\frac{d^2\ddot{u}}{dt^2})\dot{u} + \frac{\partial}{\partial \dot{u}}(\frac{d^2\ddot{u}}{dt^2})\ddot{u})\dot{u} + \frac{\partial}{\partial \dot{u}}(\frac{\partial}{\partial u}(\frac{d^2\ddot{u}}{dt^2})\dot{u} + \frac{\partial}{\partial \dot{u}}(\frac{d^2\ddot{u}}{dt^2})\ddot{u})\ddot{u}]\dot{u} \\ & + \frac{\partial}{\partial \dot{u}}[\frac{\partial}{\partial u}(\frac{\partial}{\partial u}(\frac{d^2\ddot{u}}{dt^2})\dot{u} + \frac{\partial}{\partial \dot{u}}(\frac{d^2\ddot{u}}{dt^2})\ddot{u})\dot{u} + \frac{\partial}{\partial \dot{u}}(\frac{\partial}{\partial u}(\frac{d^2\ddot{u}}{dt^2})\dot{u} + \frac{\partial}{\partial \dot{u}}(\frac{d^2\ddot{u}}{dt^2})\ddot{u})\ddot{u}]\ddot{u}\end{aligned}$$

Nach Einsetzen der 2.Ableitung:

$$\begin{aligned}\frac{d^5\ddot{u}}{dt^5} = {} & \frac{\partial}{\partial u}[\frac{\partial}{\partial u}(\frac{\partial}{\partial u}\lceil\frac{\partial}{\partial u}(\frac{d^1\ddot{u}}{dt^1})\dot{u} + \frac{\partial}{\partial \dot{u}}(\frac{d^1\ddot{u}}{dt^1})\ddot{u}\rceil\dot{u} + \frac{\partial}{\partial \dot{u}}\lceil\frac{\partial}{\partial u}(\frac{d^1\ddot{u}}{dt^1})\dot{u} + \frac{\partial}{\partial \dot{u}}(\frac{d^1\ddot{u}}{dt^1})\ddot{u}\rceil\ddot{u})\dot{u} \\ & + \frac{\partial}{\partial \dot{u}}(\frac{\partial}{\partial u}\lceil\frac{\partial}{\partial u}(\frac{d^1\ddot{u}}{dt^1})\dot{u} + \frac{\partial}{\partial \dot{u}}(\frac{d^1\ddot{u}}{dt^1})\ddot{u}\rceil\dot{u} + \frac{\partial}{\partial \dot{u}}\lceil\frac{\partial}{\partial u}(\frac{d^1\ddot{u}}{dt^1})\dot{u} + \frac{\partial}{\partial \dot{u}}(\frac{d^1\ddot{u}}{dt^1})\ddot{u}\rceil\ddot{u})\ddot{u}]\dot{u} \\ & + \frac{\partial}{\partial \dot{u}}[\frac{\partial}{\partial u}(\frac{\partial}{\partial u}\lceil\frac{\partial}{\partial u}(\frac{d^1\ddot{u}}{dt^1})\dot{u} + \frac{\partial}{\partial \dot{u}}(\frac{d^1\ddot{u}}{dt^1})\ddot{u}\rceil\dot{u} + \frac{\partial}{\partial \dot{u}}\lceil\frac{\partial}{\partial u}(\frac{d^1\ddot{u}}{dt^1})\dot{u} + \frac{\partial}{\partial \dot{u}}(\frac{d^1\ddot{u}}{dt^1})\ddot{u}\rceil\ddot{u})\dot{u} \\ & + \frac{\partial}{\partial \dot{u}}(\frac{\partial}{\partial u}\lceil\frac{\partial}{\partial u}(\frac{d^1\ddot{u}}{dt^1})\dot{u} + \frac{\partial}{\partial \dot{u}}(\frac{d^1\ddot{u}}{dt^1})\ddot{u}\rceil\dot{u} + \frac{\partial}{\partial \dot{u}}\lceil\frac{\partial}{\partial u}(\frac{d^1\ddot{u}}{dt^1})\dot{u} + \frac{\partial}{\partial \dot{u}}(\frac{d^1\ddot{u}}{dt^1})\ddot{u}\rceil\ddot{u})\ddot{u}]\ddot{u}\end{aligned}$$

Nach Einsetzen der 1.Ableitung:

$$\begin{aligned}\frac{d^5\ddot{u}}{dt^5} = {} & \frac{\partial}{\partial u}[\frac{\partial}{\partial u}(\frac{\partial}{\partial u}\lceil\frac{\partial}{\partial u}\langle\frac{\partial\ddot{u}}{\partial u}\dot{u} + \frac{\partial\ddot{u}}{\partial \dot{u}}\ddot{u}\rangle\dot{u} + \frac{\partial}{\partial \dot{u}}\langle\frac{\partial\ddot{u}}{\partial u}\dot{u} + \frac{\partial\ddot{u}}{\partial \dot{u}}\ddot{u}\rangle\ddot{u}\rceil\dot{u} + \frac{\partial}{\partial \dot{u}}\lceil\frac{\partial}{\partial u}\langle\frac{\partial\ddot{u}}{\partial u}\dot{u} + \frac{\partial\ddot{u}}{\partial \dot{u}}\ddot{u}\rangle\dot{u} \\ & + \frac{\partial}{\partial \dot{u}}\langle\frac{\partial\ddot{u}}{\partial u}\dot{u} + \frac{\partial\ddot{u}}{\partial \dot{u}}\ddot{u}\rangle\ddot{u}\rceil\ddot{u})\dot{u} + \frac{\partial}{\partial \dot{u}}(\frac{\partial}{\partial u}\lceil\frac{\partial}{\partial u}\langle\frac{\partial\ddot{u}}{\partial u}\dot{u} + \frac{\partial\ddot{u}}{\partial \, dotu}\ddot{u}\rangle\dot{u} + \frac{\partial}{\partial \dot{u}}\langle\frac{\partial\ddot{u}}{\partial u}\dot{u} + \frac{\partial\ddot{u}}{\partial \dot{u}}\ddot{u}\rangle\ddot{u}\rceil\dot{u} \\ & + \frac{\partial}{\partial \dot{u}}\lceil\frac{\partial}{\partial u}\langle\frac{\partial\ddot{u}}{\partial u}\dot{u} + \frac{\partial\ddot{u}}{\partial \dot{u}}\ddot{u}\rangle\dot{u} + \frac{\partial}{\partial \dot{u}}\langle\frac{\partial\ddot{u}}{\partial u}\dot{u} + \frac{\partial\ddot{u}}{\partial \dot{u}}\ddot{u}\rangle\ddot{u}\rceil\ddot{u})\ddot{u}]\dot{u} \\ & + \frac{\partial}{\partial \dot{u}}[\frac{\partial}{\partial u}(\frac{\partial}{\partial u}\lceil\frac{\partial}{\partial u}\langle\frac{\partial\ddot{u}}{\partial u}\dot{u} + \frac{\partial\ddot{u}}{\partial \dot{u}}\ddot{u}\rangle\dot{u} + \frac{\partial}{\partial \dot{u}}\langle\frac{\partial\ddot{u}}{\partial u}\dot{u} + \frac{\partial\ddot{u}}{\partial \dot{u}}\ddot{u}\rangle\ddot{u}\rceil\dot{u} + \frac{\partial}{\partial \dot{u}}\lceil\frac{\partial}{\partial u}\langle\frac{\partial\ddot{u}}{\partial u}\dot{u} + \frac{\partial\ddot{u}}{\partial \dot{u}}\ddot{u}\rangle\dot{u} \\ & + \frac{\partial}{\partial \dot{u}}\langle\frac{\partial\ddot{u}}{\partial u}\dot{u} + \frac{\partial\ddot{u}}{\partial \dot{u}}\ddot{u}\rangle\ddot{u}\rceil\ddot{u})\dot{u} + \frac{\partial}{\partial \dot{u}}(\frac{\partial}{\partial u}\lceil\frac{\partial}{\partial u}\langle\frac{\partial\ddot{u}}{\partial u}\dot{u} + \frac{\partial\ddot{u}}{\partial \dot{u}}\ddot{u}\rangle\dot{u} + \frac{\partial}{\partial \dot{u}}\langle\frac{\partial\ddot{u}}{\partial u}\dot{u} + \frac{\partial\ddot{u}}{\partial \dot{u}}\ddot{u}\rangle\ddot{u}\rceil\dot{u} \\ & + \frac{\partial}{\partial \dot{u}}\lceil\frac{\partial}{\partial u}\langle\frac{\partial\ddot{u}}{\partial u}\dot{u} + \frac{\partial\ddot{u}}{\partial \dot{u}}\ddot{u}\rangle\dot{u} + \frac{\partial}{\partial \dot{u}}\langle\frac{\partial\ddot{u}}{\partial u}\dot{u} + \frac{\partial\ddot{u}}{\partial \dot{u}}\ddot{u}\rangle\ddot{u}\rceil\ddot{u})\ddot{u}]\ddot{u}\end{aligned}$$

Hier kann nicht weiter rücksubstituiert werden, da die 2. Zeitableitung durch die Differentialgleichung ausgedrückt wird. Aus dieser Darstellung lassen sich einige Informationen gewinnen:

- Die Anzahl der Terme einer rücksubstituierten Darstellung nimmt um $2^{Ordnung+1}$ zu, d.h. um die jeweiligen Ordnungen zu bestimmen, wird die folgende Anzahl Terme benötigt:
  - Ordnung 1 : $2^1 = 2$ Terme
  - Ordnung 2 : $2^2 = 4$ Terme
  - Ordnung 3 : $2^3 = 8$ Terme
  - Ordnung n : 2^n Terme
- Um die vollständige Ableitung in Form einer bestimmten rücksubstituierten Ableitung einer gewissen Ordnung zu bestimmen, muss die Reihenfolge der partiellen Ableitungen für jeden der $2^{Ordnung+1}$ Terme bekannt sein.

Um einen Reihenterm einer bestimmten Ordnung zu berechnen, müssen alle vorherigen Ableitungsterme ermittelt werden. Werden rücksubstituierte Reihenkoeffizienten gebildet, so geschieht dies durch aufeinander folgendes partielles Differenzieren des vorangegangen Terms. Die Entwicklung ist in Abbildung B.1 dargestellt.

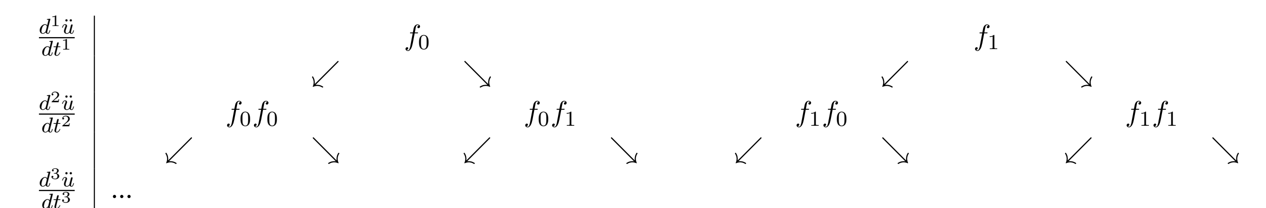


Tabelle B.1: Entwicklung höherer, rücksubstituierter Ableitungsterme

wobei

$$f_0 = \frac{\partial \ddot{u}}{\partial u}\dot{u}$$

$$f_1 = \frac{\partial \ddot{u}}{\partial \dot{u}}\ddot{u}$$

gesetzt wurde. Außerdem werden gemischt partielle Ableitungen wie folgt abgekürzt geschrieben:

$$f_0 f_1 = \frac{\partial \frac{\partial \ddot{u}}{\partial \dot{u}}\ddot{u}}{\partial u}\dot{u}$$

Aus Tabelle B.1 kann das Bildungsgesetz zu Bestimmung eines bestimmten Reihenkoeffizienten abgelesen werden. Beispielsweise sind zur Berechnung von $\frac{d^3\ddot{u}}{dt^3}$ folgende partielle Ableitungen zu bilden:

$$\frac{d^3\ddot{u}}{dt^3} = f_0f_0f_0 + f_0f_0f_1 + f_0f_1f_0 + f_0f_1f_1 + f_1f_0f_0 + f_1f_0f_1 + f_1f_1f_0 + f_1f_1f_1$$

Betrachtet man die Indices genauer, so entspricht dies exakt der Binärkodierung, wie sie bei der digitalen Verarbeitung von Information verwendet wird. Grundlage eines jeden Binärcodes ist ein so genanntes Binäres System, d.h. ein System, in dem nur ausschließlich zwei Zustände herrschen (hier 0 oder 1). Bei der Zahlendarstellung im Dualsystem werden die Ziffern z_i wie im gewöhnlich verwendeten Dezimalsystem ohne Trennzeichen hintereinander geschrieben, ihr Stellenwert entspricht allerdings der zur Stelle passenden Zweierpotenz und nicht der Zehnerpotenz. Die höchstwertige Stelle mit dem Wert z_m wird ganz links und die niederwertigen Stellen mit den Werten z_{m-1} bisz_0 in absteigender Reihenfolge rechts davon geschrieben.

$$z_m z_{m-1} ... z_0$$

wobei

$$z \in N, z_i \in \{0, 1\}$$

Die dezimale Darstellung kann durch folgenden Zusammenhang bestimmt werden:

$$Z = \sum_{i=0}^{m} z_i * 2^i$$

In Tabelle B.2 ist die Binärcodierung für den Fall $\frac{d^3\ddot{u}}{dt^3}$ dargestellt. Hierbei ist zu sehen, dass die Anzahl der zu berechnenden Terme mit jeder hinzukommenden Ordnung um eine Spalte (im Binärsystem) steigt.

Binär	Dezimal	Folge aufeinanderfolgender partiellen Ableitungen
000	0	$f_0 f_0 f_0$
001	1	$f_0 f_0 f_1$
010	2	$f_0 f_1 f_0$
011	3	$f_0 f_1 f_1$
100	4	$f_1 f_0 f_0$
101	5	$f_1 f_0 f_1$
110	6	$f_1 f_1 f_0$
111	7	$f_1 f_1 f_1$

Tabelle B.2: Folge aufeinanderfolgender partieller Ableitungen in binärer und dezimaler Darstellung

Die Berechnung einzelner Reihenkoeffizienten kann mit Hilfe symbolischer Rechnung bestimmt werden. Durch die vorher gezeigte Entwicklung ist die Berechnung der Reihenkoeffizienten parallelisierbar, d.h. die jeweils zu berechnenden Koeffizienten können unter Ausnutzung von Mehrkernprozessoren bestimmt werden. Dadurch können in kürzerer Zeit Terme höherer Ordnung berechnet werden. Diese wiederum sind bereits rücksubstituiert und können ebenfalls parallel ausgewertet werden, was einen Zuwachs der Abarbeitungsgeschwindigeit auf Mehrkernprozessoren verspricht. Dies soll am Beispiel der Differentialgleichung des Duffing-Oszillators demonstriert werden. Die Differentialgleichung des ungedämpften Duffing-Oszillators ist wie folgt definiert:

$$\ddot{u} = -\omega^2 \dot{u} - \varepsilon \dot{u}^3$$

wobei:

- $\ddot{u}$ die zweite Ableitung nach der Zeit ist.
- $\dot{u}$ die erste Ableitung nach der Zeit ist.
- ω und ε konstante Faktoren sind.

Das folgende Programm berechnet Reihenkoeffizienten bis zu $\frac{d^n\ddot{u}}{dt^n}$ mit Hilfe von SymbolicC++ und gibt das jeweilige Ergebnis in die Standardausgabe aus. Die Ordnung der Ableitungsterme kann durch Angabe eines Kommandozeilenparameters angegeben werden.

```
1 #include "symbolicc++.h"
2 int main(int iargc, char *argv[])
3 {
4     int iN = 10;
5     if(iargc >= 1)
6         iN =atoi(argv[1]);
7     if(iN < 0)
8     {
9       std::cerr << "negative number provided!\n";
10      exit(-1);
11    }
12    Symbolic omega("omega"),u("u"),eps("eps"),t("t");
13    u=u[t];
14    Symbolic F;
15    F = -((omega^(2))*u) - (eps*(u^(3)));
16    Symbolic     dF = df(F ,t);
17    std::cout << dF << std::endl;
18    for(int i = 0; i < iN; i++)
19    {
20        Symbolic ddF = df(dF,t)[(df(df(u,t),t)) == F];
21        std::cout << ddF << std::endl;
22        dF = ddF;
23    }
24    return 0;
25 }
```

Listing B.1: Beispielprogramm zum Erzeugen höherer Ableitungen

Folgend sind die höheren Ableitungen bis zur 5. Ordnung angegeben. Mit dem oben gezeigten Programm können prinzipiell beliebig hohe Ableitungsterme berechnet werden.

$$\frac{d^3\ddot{u}}{dt^3} = -3\dot{u}\varepsilon u^2 - \dot{u}\omega^2$$
$$\frac{d^4\ddot{u}}{dt^4} = -6\dot{u}^2\varepsilon u + \omega^4 u + 4\varepsilon\omega^2 u^3 + 3\varepsilon^2 u^5$$
$$\frac{d^5\ddot{u}}{dt^5} = -6\dot{u}^3\varepsilon + \dot{u}\omega^4 + 24\dot{u}\varepsilon\omega^2 u^2 + 27\dot{u}\varepsilon^2 u^4$$
$$\frac{d^6\ddot{u}}{dt^6} = ...$$

Diese sollen im Folgenden auf ihre Richtigkeit überprüft werden, indem sie mit der numerisch berechneten Lösung des Potenzreihenintegrators verglichen werden. Außerdem soll gezeigt werden, wie ein bestimmter Ableitungsterm berechnet wird. Das daraus resultierende Ergebnis wird ebenfalls mit dem Potenzreihenintegrator verglichen.

Anhang C

Höhere Ableitungen der Keplerbahn

Die folgenden Variablen werden als gegeben vorausgesetzt:

$$\vec{r} = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix}; \quad r = \sqrt{x^2(t) + y^2(t) + z^2(t)};$$

$$\dot{\vec{r}} = \begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{z}(t) \end{pmatrix}; \quad \dot{r} = \frac{dr}{dt} = \frac{2z\left(t\right)\left(\frac{d}{dt}z\left(t\right)\right) + 2y\left(t\right)\left(\frac{d}{dt}y\left(t\right)\right) + 2x\left(t\right)\left(\frac{d}{dt}x\left(t\right)\right)}{2\sqrt{z^2\left(t\right) + y^2\left(t\right) + x^2\left(t\right)}} = \frac{[\vec{r}\dot{\vec{r}}]}{r};$$

$$\mu = GM_\oplus;$$

dabei ist:

- $\vec{r}$ der Ortsvektor im Inertialsystem und r die geometrische Distanz.
- $\dot{\vec{r}}$ der Vektor der Geschwindigkeitskomponenten und $\dot{r}$ die Radialgeschwindigkeit.
- μ die Gravitationskonstante G multipliziert mit der Masse der Erde $M_\oplus$.
- t die Zeit.

Hinweise zur Notation
Der Ausdruck $[\vec{a}\vec{b}]$ entspricht dem Skalarprodukt zweier Vektoren, d.h. $[\vec{a}\vec{b}] := a_0b_0 + ... + a_nb_n$.
Wobei die Vektoren $\vec{a}$ und $\vec{b}$ aus $n + 1$ Komponenten bestehen.
Diese Notation ist insbesondere zur Kontrolle der Dimension größerer Ausdrücke hilfreich.

$$\frac{d^0\vec{r}}{dt^0} = \vec{r}$$

$$\frac{d^1\vec{r}}{dt^1} = \dot{\vec{r}}$$

$$\frac{d^2\vec{r}}{dt^2} = -\mu\frac{\vec{r}}{r^3}$$

$$\frac{d^3\vec{r}}{dt^3} = -\mu[\frac{\dot{\vec{r}}}{r^3} - \frac{3\vec{r}[\vec{r}\dot{\vec{r}}]}{r^5}]$$

$$\frac{d^4\vec{r}}{dt^4} = \mu[\frac{3\vec{r}([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])}{r^5} - \frac{15\vec{r}[\vec{r}\dot{\vec{r}}]^2}{r^7} + \frac{6\dot{\vec{r}}[\vec{r}\dot{\vec{r}}]}{r^5} - \frac{1}{r^3}\ddot{\vec{r}}]$$

$$\begin{aligned}\frac{d^5\vec{r}}{dt^5} = \mu[&\frac{3\vec{r}([\vec{r}\frac{d^3\vec{r}}{dt^3}] + 3[\dot{\vec{r}}\ddot{\vec{r}}])}{r^5} - \frac{45\vec{r}[\vec{r}\dot{\vec{r}}](\vec{r}\ddot{\vec{r}} + [\dot{\vec{r}}\dot{\vec{r}}])}{r^7} + \frac{9\dot{\vec{r}}([\vec{r}\ddot{\vec{r}}] + [\dot{r}]^2)}{r^5} + \frac{105\vec{r}[\vec{r}\dot{\vec{r}}]^3}{r^9} - \frac{45\dot{\vec{r}}[\vec{r}\dot{\vec{r}}]^2}{r^7} + \frac{9\ddot{\vec{r}}[\vec{r}\dot{\vec{r}}]}{r^5}\\ &- \frac{\frac{d^3\vec{r}}{dt^3}}{r^3}]\end{aligned}$$

$$\begin{aligned}\frac{d^6\vec{r}}{dt^6} = \mu[&\frac{3\vec{r}([\vec{r}\frac{d^4\vec{r}}{dt^4}] + 4[\dot{\vec{r}}\frac{d^3\vec{r}}{dt^3}] + 3[\ddot{\vec{r}}\ddot{\vec{r}}])}{r^5} - \frac{60\vec{r}[\vec{r}\dot{\vec{r}}]([\vec{r}\frac{d^3\vec{r}}{dt^3}] + [3\dot{\vec{r}}\ddot{\vec{r}}])}{r^7} + \frac{12\dot{\vec{r}}([\vec{r}\frac{d^3\vec{r}}{dt^3}] + [3\dot{\vec{r}}\ddot{\vec{r}}])}{r^5}\\ &+ \frac{630\vec{r}[\vec{r}\dot{\vec{r}}]^2([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])}{r^9} - \frac{180\dot{\vec{r}}[\vec{r}\dot{\vec{r}}]([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])}{r^7} + \frac{18\ddot{\vec{r}}([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])}{r^5} - \frac{945\vec{r}[\vec{r}\dot{\vec{r}}]^4}{r^{11}} + \frac{420\dot{\vec{r}}[\vec{r}\dot{\vec{r}}]^3}{r^9}\\ &- \frac{90\ddot{\vec{r}}[\vec{r}\dot{\vec{r}}]^2}{r^7} + \frac{12\frac{d^3\vec{r}}{dt^3}[\vec{r}\dot{\vec{r}}]}{r^5} - \frac{\frac{d^4\vec{r}}{dt^4}}{r^3} - \frac{45\vec{r}([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])^2)}{r^7}]\end{aligned}$$

$$\begin{aligned}\frac{d^7\vec{r}}{dt^7} = \mu[&\frac{3\vec{r}([\vec{r}\frac{d^5\vec{r}}{dt^5}] + [5\dot{\vec{r}}\frac{d^4\vec{r}}{dt^4}] + [10\ddot{\vec{r}}\frac{d^3\vec{r}}{dt^3}])}{r^5} - \frac{\frac{d^5\vec{r}}{dt^5}}{r^3} - \frac{75\vec{r}[\vec{r}\dot{\vec{r}}]([\vec{r}\frac{d^4\vec{r}}{dt^4}] + 4[\dot{\vec{r}}\frac{d^3\vec{r}}{dt^3}] + 3[\ddot{\vec{r}}\ddot{\vec{r}}])}{r^7}\\ &+ \frac{15\dot{\vec{r}}([\vec{r}\frac{d^4\vec{r}}{dt^4}] + 4[\dot{\vec{r}}\frac{d^3\vec{r}}{dt^3}] + 3[\ddot{\vec{r}}\ddot{\vec{r}}])}{r^5} + \frac{15[\vec{r}\dot{\vec{r}}]\frac{d^4\vec{r}}{dt^4}}{r^5} - \frac{150\vec{r}([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])([\vec{r}\frac{d^3\vec{r}}{dt^3}] + 3[\dot{\vec{r}}\ddot{\vec{r}}])}{r^7}\\ &+ \frac{30\ddot{\vec{r}}([\vec{r}\frac{d^3\vec{r}}{dt^3}] + 3[\dot{\vec{r}}\ddot{\vec{r}}])}{r^5} + \frac{1050\vec{r}[\vec{r}\dot{\vec{r}}]^2([\vec{r}\frac{d^3\vec{r}}{dt^3}] + 3[\dot{\vec{r}}\ddot{\vec{r}}])}{r^9} - \frac{300\dot{\vec{r}}[\vec{r}\dot{\vec{r}}]([\ vecr\frac{d^3\vec{r}}{dt^3}] + 3[\dot{\vec{r}}\ddot{\vec{r}}])}{r^7}\\ &+ \frac{30\frac{d^3\vec{r}}{dt^3}([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])}{r^5} - \frac{150\frac{d^3\vec{r}}{dt^3}[\vec{r}\dot{\vec{r}}]^2}{r^7} + \frac{1575\vec{r}[\vec{r}\dot{\vec{r}}]([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])^2}{r^9} - \frac{225\dot{\vec{r}}([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])^2}{r^7}\\ &- \frac{450\ddot{\vec{r}}[\vec{r}\dot{\vec{r}}]([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])}{r^7} - \frac{9450\vec{r}[\vec{r}\dot{\vec{r}}]^3([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])}{r^{11}} + \frac{3150\dot{\vec{r}}[\vec{r}\dot{\vec{r}}]^2([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])}{r^9} + \frac{1050\ddot{\vec{r}}[\vec{r}\dot{\vec{r}}]^3}{r^9}\\ &+ \frac{10395\vec{r}[\vec{r}\dot{\vec{r}}]^5}{r^{13}} - \frac{4725\dot{\vec{r}}[\vec{r}\dot{\vec{r}}]^4}{r^{11}}]\end{aligned}$$

$$
\begin{aligned}
\frac{d^8\vec{r}}{dt^8} = \mu[& \frac{3([\vec{r}\frac{d^6\vec{r}}{dt^6}] + 6[\dot{\vec{r}}\frac{d^5\vec{r}}{dt^5}] + 15[\ddot{\vec{r}}\frac{d^4\vec{r}}{dt^4}] + 10[\frac{d^3\vec{r}}{dt^3}\frac{d^3\vec{r}}{dt^3}])\vec{r}}{r^5} - \frac{90[\vec{r}\dot{\vec{r}}]([\vec{r}\frac{d^5\vec{r}}{dt^5}] + 5[\dot{\vec{r}}\frac{d^4\vec{r}}{dt^4}] + 10[\ddot{\vec{r}}\frac{d^3\vec{r}}{dt^3}])\vec{r}}{r^7} \\
& - \frac{150([\vec{r}\frac{d^3\vec{r}}{dt^3}] + 3[\dot{\vec{r}}\ddot{\vec{r}}])^2)\vec{r}}{r^7} - \frac{225([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])\,([\vec{r}\frac{d^4\vec{r}}{dt^4}] + 4[\dot{\vec{r}}\frac{d^3\vec{r}}{dt^3}] + 3[\ddot{\vec{r}}\ddot{\vec{r}}])\vec{r}}{r^7} \\
& + \frac{1575[\vec{r}\dot{\vec{r}}]^2([\vec{r}\frac{d^4\vec{r}}{dt^4}] + 4[\dot{\vec{r}}\frac{d^3\vec{r}}{dt^3}] + 3[\ddot{\vec{r}}\ddot{\vec{r}}])\vec{r}}{r^9} + \frac{1575([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])^3\vec{r}}{r^9} \\
& + \frac{6300[\vec{r}\dot{\vec{r}}]([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])([\vec{r}\frac{d^3\vec{r}}{dt^3}] + 3[\dot{\vec{r}}\ddot{\vec{r}}])\vec{r}}{r^9} - \frac{18900[\vec{r}\dot{\vec{r}}]^3([\vec{r}\frac{d^3\vec{r}}{dt^3}] + 3[\dot{\vec{r}}\ddot{\vec{r}}])\vec{r}}{r^{11}} \\
& - \frac{42525[\dot{\vec{r}}\dot{\vec{r}}]^2([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])^2\vec{r}}{r^{11}} - \frac{135135[\vec{r}\dot{\vec{r}}]^6\vec{r}}{r^{15}} + \frac{155925[\vec{r}\dot{\vec{r}}]^4([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])\vec{r}}{r^{13}} \\
& + \frac{18([\vec{r}\frac{d^5\vec{r}}{dt^5}] + 5[\dot{\vec{r}}\frac{d^4\vec{r}}{dt^4}] + 10[\ddot{\vec{r}}\frac{d^3\vec{r}}{dt^3}])\dot{\vec{r}}}{r^5} - \frac{450[\dot{\vec{r}}\dot{\vec{r}}]([\dot{\vec{r}}\frac{d^4\vec{r}}{dt^4}] + 4[\dot{\vec{r}}\frac{d^3\vec{r}}{dt^3}] + 3[\ddot{\vec{r}}\frac{d^3\vec{r}}{dt^3}])\dot{\vec{r}}}{r^7} \\
& - \frac{900([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])([\vec{r}\frac{d^3\vec{r}}{dt^3}] + 3[\dot{\vec{r}}\ddot{\vec{r}}])\dot{\vec{r}}}{r^7} + \frac{6300[\vec{r}\dot{\vec{r}}]^2([\vec{r}\frac{d^3\vec{r}}{dt^3}] + 3[\dot{\vec{r}}\ddot{\vec{r}}])\dot{\vec{r}}}{r^9} \\
& + \frac{9450[\vec{r}\dot{\vec{r}}]([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])^2\dot{\vec{r}}}{r^9} - \frac{56700[\vec{r}\dot{\vec{r}}]^3([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])\dot{\vec{r}}}{r^{11}} + \frac{62370[\vec{r}\dot{\vec{r}}]^5\dot{\vec{r}}}{r^{13}} \\
& + \frac{45([\vec{r}\frac{d^4\vec{r}}{dt^4}] + 4[\dot{\vec{r}}\frac{d^3\vec{r}}{dt^3}] + 3[\ddot{\vec{r}}\ddot{\vec{r}}])\ddot{\vec{r}}}{r^5} - \frac{675([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])^2\ddot{\vec{r}}}{r^7} \\
& - \frac{900[\vec{r}\dot{\vec{r}}]([\vec{r}\frac{d^3\vec{r}}{dt^3}] + 3[\dot{\vec{r}}\ddot{\vec{r}}])\ddot{\vec{r}}}{r^7} + \frac{9450[\vec{r}\dot{\vec{r}}]^2([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])\ddot{\vec{r}}}{r^9} - \frac{14175[\vec{r}\dot{\vec{r}}]^4\ddot{\vec{r}}}{r^{11}} \\
& + \frac{60([\vec{r}\frac{d^3\vec{r}}{dt^3}] + 3[\dot{\vec{r}}\ddot{\vec{r}}])\frac{d^3\vec{r}}{dt^3}}{r^5} - \frac{900[\vec{r}\dot{\vec{r}}]([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])\frac{d^3\vec{r}}{dt^3}}{r^7} + \frac{2100[\vec{r}\dot{\vec{r}}]^3\frac{d^3\vec{r}}{dt^3}}{r^9} \\
& + \frac{45([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])\frac{d^4\vec{r}}{dt^4}}{r^5} - \frac{225[\vec{r}\dot{\vec{r}}]^2\frac{d^4\vec{r}}{dt^4}}{r^7} + \frac{18[\vec{r}\dot{\vec{r}}]\frac{d^5\vec{r}}{dt^5}}{r^5} - \frac{\frac{d^6\vec{r}}{dt^6}}{r^3}]
\end{aligned}
$$

$$
\begin{aligned}
\frac{d^9\vec r}{dt^9} = \mu[& \frac{3([\vec r\frac{d^7\vec r}{dt^7}] + 7[\dot{\vec r}\frac{d^6\vec r}{dt^6}] + 21[\ddot{\vec r}\frac{d^5\vec r}{dt^5}] + 35[\frac{d^3\vec r}{dt^3}\frac{d^4\vec r}{dt^4}])\vec r}{r^5} \\
& - \frac{105[\vec r\dot{\vec r}](6[\dot{\vec r}\frac{d^5\vec r}{dt^5}] + [\vec r\frac{d^6\vec r}{dt^6}] + 10[\frac{d^3\vec r}{dt^3}\frac{d^3\vec r}{dt^3}] + 15[\ddot{\vec r}\frac{d^4\vec r}{dt^4}])\vec r}{r^7} \\
& - \frac{315([\vec r\ddot{\vec r}] + [\dot{\vec r}\dot{\vec r}])(5[\dot{\vec r}\frac{d^4\vec r}{dt^4}] + [\vec r\frac{d^5\vec r}{dt^5}] + 10[\ddot{\vec r}\frac{d^3\vec r}{dt^3}])\vec r}{r^7} \\
& - \frac{525(3[\dot{\vec r}\ddot{\vec r}] + [\vec r\frac{d^3\vec r}{dt^3}])(4[\dot{\vec r}\frac{d^3\vec r}{dt^3}] + [\vec r\frac{d^4\vec r}{dt^4}] + 3[\ddot{\vec r}\ddot{\vec r}])\vec r}{r^7} \\
& + \frac{2205[\vec r\dot{\vec r}]^2(5[\dot{\vec r}\frac{d^4\vec r}{dt^4}] + [\vec r\frac{d^5\vec r}{dt^5}] + 10[\ddot{\vec r}\frac{d^3\vec r}{dt^3}])\vec r}{r^9} + \frac{7350[\vec r\dot{\vec r}](3[\dot{\vec r}\ddot{\vec r}] + [\vec r\frac{d^3\vec r}{dt^3}])^2\vec r}{r^9} \\
& + \frac{11025[\vec r\dot{\vec r}]([\vec r\ddot{\vec r}] + [\dot{\vec r}\dot{\vec r}])(4[\dot{\vec r}\frac{d^3\vec r}{dt^3}] + [\vec r\frac{d^4\vec r}{dt^4}] + 3[\ddot{\vec r}\ddot{\vec r}])\vec r}{r^9} + \frac{11025([\vec r\ddot{\vec r}] + [\dot{\vec r}\dot{\vec r}])^2(3[\dot{\vec r}\ddot{\vec r}] + [\vec r\frac{d^3\vec r}{dt^3}])\vec r}{r^9} \\
& - \frac{33075[\vec r\dot{\vec r}]^3(4[\dot{\vec r}\frac{d^3\vec r}{dt^3}] + [\vec r\frac{d^4\vec r}{dt^4}] + 3[\ddot{\vec r}\ddot{\vec r}])\vec r}{r^{11}} - \frac{99225[\vec r\dot{\vec r}]([\vec r\ddot{\vec r}] + [\dot{\vec r}\dot{\vec r}])^3\vec r}{r^{11}} \\
& - \frac{198450[\vec r\dot{\vec r}]^2([\vec r\ddot{\vec r}] + [\dot{\vec r}\dot{\vec r}])((3[\dot{\vec r}\ddot{\vec r}] + [\vec r\frac{d^3\vec r}{dt^3}])\vec r}{r^{11}} + \frac{363825[\vec r\dot{\vec r}]^4(3[\dot{\vec r}\ddot{\vec r}] + [\vec r\frac{d^3\vec r}{dt^3}])\vec r}{r^{13}} \\
& + \frac{1091475[\vec r\dot{\vec r}]^3(([\vec r\ddot{\vec r}] + [\dot{\vec r}\dot{\vec r}])^2)\vec r}{r^{13}} - \frac{2837835[\vec r\dot{\vec r}]^5([\vec r\ddot{\vec r}] + [\dot{\vec r}\dot{\vec r}])\vec r}{r^{15}} + \frac{2027025[\vec r\dot{\vec r}]^7\vec r}{r^{17}} \\
& + \frac{21(6[\dot{\vec r}\frac{d^5\vec r}{dt^5}] + [\vec r\frac{d^6\vec r}{dt^6}] + 10[\frac{d^3\vec r}{dt^3}\frac{d^3\vec r}{dt^3}] + 15[\ddot{\vec r}\frac{d^4\vec r}{dt^4}])\dot{\vec r}}{r^5} - \frac{630[\vec r\dot{\vec r}](5[\dot{\vec r}\frac{d^4\vec r}{dt^4}] + [\vec r\frac{d^5\vec r}{dt^5}] + 10[\ddot{\vec r}\frac{d^3\vec r}{dt^3}])\dot{\vec r}}{r^7} \\
& - \frac{1050((3[\dot{\vec r}\ddot{\vec r}] + [\vec r\frac{d^3\vec r}{dt^3}])^2)\dot{\vec r}}{r^7} - \frac{1575([\vec r\ddot{\vec r}] + [\dot{\vec r}\dot{\vec r}])(4[\dot{\vec r}\frac{d^3\vec r}{dt^3}] + [\vec r\frac{d^4\vec r}{dt^4}] + 3[\ddot{\vec r}\ddot{\vec r}])\dot{\vec r}}{r^7} \\
& + \frac{11025([\vec r\ddot{\vec r}] + [\dot{\vec r}\dot{\vec r}])^3)\dot{\vec r}}{r^9} + \frac{11025[\vec r\dot{\vec r}]^2(4[\dot{\vec r}\frac{d^3\vec r}{dt^3}] + [\vec r\frac{d^4\vec r}{dt^4}] + 3[\ddot{\vec r}\ddot{\vec r}])\dot{\vec r}}{r^9} \\
& + \frac{44100[\vec r\dot{\vec r}]([\vec r\ddot{\vec r}] + [\dot{\vec r}\dot{\vec r}])(3[\dot{\vec r}\ddot{\vec r}] + [\vec r\frac{d^3\vec r}{dt^3}]\dot{\vec r}}{r^9} - \frac{66150[\vec r\dot{\vec r}]^3(6[\dot{\vec r}\ddot{\vec r}] + 2[\vec r\frac{d^3\vec r}{dt^3}])\dot{\vec r}}{r^{11}} \\
& - \frac{297675[\vec r\dot{\vec r}]^2([\vec r\ddot{\vec r}] + [\dot{\vec r}\dot{\vec r}])^2)\dot{\vec r}}{r^{11}} + \frac{1091475[\vec r\dot{\vec r}]^4([\vec r\ddot{\vec r}] + [\dot{\vec r}\dot{\vec r}])\dot{\vec r}}{r^{13}} - \frac{945945[\vec r\dot{\vec r}]^6\dot{\vec r}}{r^{15}} \\
& + \frac{63(5[\dot{\vec r}\frac{d^4\vec r}{dt^4}] + [\vec r\frac{d^5\vec r}{dt^5}] + 10[\ddot{\vec r}\frac{d^3\vec r}{dt^3}])\frac{d^2\vec r}{dt^2}}{r^5} - \frac{1575[\vec r\dot{\vec r}](4[\dot{\vec r}\frac{d^3\vec r}{dt^3}] + [\vec r\frac{d^4\vec r}{dt^4}] + 3[\ddot{\vec r}\ddot{\vec r}])\frac{d^2\vec r}{dt^2}}{r^7} \\
& - \frac{3150([\vec r\ddot{\vec r}] + [\dot{\vec r}\dot{\vec r}])(3[\dot{\vec r}\ddot{\vec r}] + [\vec r\frac{d^3\vec r}{dt^3}])\frac{d^2\vec r}{dt^2}}{r^7} + \frac{11025[\vec r\dot{\vec r}]^2(6[\dot{\vec r}\ddot{\vec r}] + 2[\vec r\frac{d^3\vec r}{dt^3}])\frac{d^2\vec r}{dt^2}}{r^9} \\
& + \frac{33075[\vec r\dot{\vec r}]([\vec r\ddot{\vec r}] + [\dot{\vec r}\dot{\vec r}])^2\frac{d^2\vec r}{dt^2}}{r^9} - \frac{198450[\vec r\dot{\vec r}]^3([\vec r\ddot{\vec r}] + [\dot{\vec r}\dot{\vec r}])\frac{d^2\vec r}{dt^2}}{r^{11}} + \frac{218295[\vec r\dot{\vec r}]^5\frac{d^2\vec r}{dt^2}}{r^{13}} \\
& + \frac{105(4[\dot{\vec r}\frac{d^3\vec r}{dt^3}] + [\vec r\frac{d^4\vec r}{dt^4}] + 3[\ddot{\vec r}\ddot{\vec r}])\frac{d^3\vec r}{dt^3}}{r^5} - \frac{1050[\vec r\dot{\vec r}](6[\dot{\vec r}\ddot{\vec r}] + 2[\vec r\frac{d^3\vec r}{dt^3}])\frac{d^3\vec r}{dt^3}}{r^7} \\
& - \frac{1575([\vec r\ddot{\vec r}] + [\dot{\vec r}\dot{\vec r}])^2\frac{d^3\vec r}{dt^3}}{r^7} + \frac{22050[\vec r\dot{\vec r}]^2([\vec r\ddot{\vec r}] + [\dot{\vec r}\dot{\vec r}])\frac{d^3\vec r}{dt^3}}{r^9} - \frac{33075[\vec r\dot{\vec r}]^4\frac{d^3\vec r}{dt^3}}{r^{11}} \\
& + \frac{105(3[\dot{\vec r}\ddot{\vec r}] + [\vec r\frac{d^3\vec r}{dt^3}])\frac{d^4\vec r}{dt^4}}{r^5} - \frac{1575[\vec r\dot{\vec r}]([\vec r\ddot{\vec r}] + [\dot{\vec r}\dot{\vec r}])\frac{d^4\vec r}{dt^4}}{r^7} + \frac{3675[\vec r\dot{\vec r}]^3\frac{d^4\vec r}{dt^4}}{r^9} \\
& + \frac{63([\vec r\ddot{\vec r}] + [\dot{\vec r}\dot{\vec r}])\frac{d^5\vec r}{dt^5}}{r^5} - \frac{315[\vec r\dot{\vec r}]^2\frac{d^5\vec r}{dt^5}}{r^7} + \frac{21[\vec r\dot{\vec r}]\frac{d^6\vec r}{dt^6}}{r^5} - \frac{\frac{d^7\vec r}{dt^7}}{r^3}]
\end{aligned}
$$

$$
\begin{aligned}
\frac{d^{10}\vec{r}}{dt^{10}} = \mu[&\frac{(3(8[\vec{r}\frac{d^7\vec{r}}{dt^7}] + [\vec{r}\frac{d^8\vec{r}}{dt^8}] + 35[\frac{d^4\vec{r}}{dt^4}\frac{d^4\vec{r}}{dt^4}] + 56[\frac{d^3\vec{r}}{dt^3}\frac{d^5\vec{r}}{dt^5}] + 28[\ddot{\vec{r}}\frac{d^6\vec{r}}{dt^6}])\vec{r}}{r^5} \\
&+ \frac{24(7[\dot{\vec{r}}\frac{d^6\vec{r}}{dt^6}] + [\vec{r}\frac{d^7\vec{r}}{dt^7}] + 35[\frac{d^3\vec{r}}{dt^3}\frac{d^4\vec{r}}{dt^4}] + 21[\ddot{\vec{r}}\frac{d^5\vec{r}}{dt^5}])\dot{\vec{r}}}{r^5} \\
&- \frac{120[\vec{r}\dot{\vec{r}}](7[\dot{\vec{r}}\frac{d^6\vec{r}}{dt^6}] + [\vec{r}\frac{d^7\vec{r}}{dt^7}] + 35[\frac{d^3\vec{r}}{dt^3}\frac{d^4\vec{r}}{dt^4}] + 21[\ddot{\vec{r}}\frac{d^5\vec{r}}{dt^5}])\vec{r}}{r^7} \\
&- \frac{420([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])(6[\dot{\vec{r}}\frac{d^5\vec{r}}{dt^5}] + [\vec{r}\frac{d^6\vec{r}}{dt^6}] + 10[\frac{d^3\vec{r}}{dt^3}\frac{d^3\vec{r}}{dt^3}] + 15[\ddot{\vec{r}}\frac{d^4\vec{r}}{dt^4}])\vec{r}}{r^7} \\
&+ \frac{84(6[\dot{\vec{r}}\frac{d^5\vec{r}}{dt^5}] + [\vec{r}\frac{d^6\vec{r}}{dt^6}] + 10[\frac{d^3\vec{r}}{dt^3}\frac{d^3\vec{r}}{dt^3}] + 15[\ddot{\vec{r}}\frac{d^4\vec{r}}{dt^4}])\ddot{\vec{r}}}{r^5} \\
&- \frac{840[\vec{r}\dot{\vec{r}}](6[\dot{\vec{r}}\frac{d^5\vec{r}}{dt^5}] + [\vec{r}\frac{d^6\vec{r}}{dt^6}] + 10[\frac{d^3\vec{r}}{dt^3}\frac{d^3\vec{r}}{dt^3}] + 15[\ddot{\vec{r}}\frac{d^4\vec{r}}{dt^4}])\dot{\vec{r}}}{r^7} \\
&+ \frac{2940[\vec{r}\dot{\vec{r}}]^2(6[\dot{\vec{r}}\frac{d^5\vec{r}}{dt^5}] + [\vec{r}\frac{d^6\vec{r}}{dt^6}] + 10[\frac{d^3\vec{r}}{dt^3}\frac{d^3\vec{r}}{dt^3}] + 15[\ddot{\vec{r}}\frac{d^4\vec{r}}{dt^4}])\vec{r}}{r^9} \\
&- \frac{840(3[\dot{\vec{r}}\ddot{\vec{r}}] + [\vec{r}\frac{d^3\vec{r}}{dt^3}])(5[\dot{\vec{r}}\frac{d^4\vec{r}}{dt^4}] + [\vec{r}\frac{d^5\vec{r}}{dt^5}] + 10[\ddot{\vec{r}}\frac{d^3\vec{r}}{dt^3}])\vec{r}}{r^7} \\
&- \frac{2520([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])(5[\dot{\vec{r}}\frac{d^4\vec{r}}{dt^4}] + [\vec{r}\frac{d^5\vec{r}}{dt^5}] + 10[\ddot{\vec{r}}\frac{d^3\vec{r}}{dt^3}])\dot{\vec{r}}}{r^7} \\
&+ \frac{17640[\vec{r}\dot{\vec{r}}]([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])(5[\dot{\vec{r}}\frac{d^4\vec{r}}{dt^4}] + [\vec{r}\frac{d^5\vec{r}}{dt^5}] + 10[\ddot{\vec{r}}\frac{d^3\vec{r}}{dt^3}])\vec{r}}{r^9} + \frac{168(5[\dot{\vec{r}}\frac{d^4\vec{r}}{dt^4}] + [\vec{r}\frac{d^5\vec{r}}{dt^5}] + 10[\ddot{\vec{r}}\frac{d^3\vec{r}}{dt^3}])\frac{d^3\vec{r}}{dt^3}}{r^5} \\
&- \frac{2520[\vec{r}\dot{\vec{r}}](5[\dot{\vec{r}}\frac{d^4\vec{r}}{dt^4}] + [\vec{r}\frac{d^5\vec{r}}{dt^5}] + 10[\ddot{\vec{r}}\frac{d^3\vec{r}}{dt^3}])\ddot{\vec{r}}}{r^7} + \frac{17640[\vec{r}\dot{\vec{r}}]^2(5[\dot{\vec{r}}\frac{d^4\vec{r}}{dt^4}] + [\vec{r}\frac{d^5\vec{r}}{dt^5}] + 10[\ddot{\vec{r}}\frac{d^3\vec{r}}{dt^3}])\dot{\vec{r}}}{r^9} \\
&- \frac{52920[\vec{r}\dot{\vec{r}}]^3(5[\dot{\vec{r}}\frac{d^4\vec{r}}{dt^4}] + [\vec{r}\frac{d^5\vec{r}}{dt^5}] + 10[\ddot{\vec{r}}\frac{d^3\vec{r}}{dt^3}])\vec{r}}{r^{11}} - \frac{525(4[\dot{\vec{r}}\frac{d^3\vec{r}}{dt^3}] + [\vec{r}\frac{d^4\vec{r}}{dt^4}] + 3[\ddot{\vec{r}}\ddot{\vec{r}}])^2\vec{r}}{r^7} \\
&- \frac{4200(3[\dot{\vec{r}}\ddot{\vec{r}}] + [\vec{r}\frac{d^3\vec{r}}{dt^3}])(4[\dot{\vec{r}}\frac{d^3\vec{r}}{dt^3}] + [\vec{r}\frac{d^4\vec{r}}{dt^4}] + 3[\ddot{\vec{r}}\ddot{\vec{r}}])\dot{\vec{r}}}{r^7} \\
&+ \frac{29400[\vec{r}\dot{\vec{r}}](3[\dot{\vec{r}}\ddot{\vec{r}}] + [\vec{r}\frac{d^3\vec{r}}{dt^3}])(4[\dot{\vec{r}}\frac{d^3\vec{r}}{dt^3}] + [\vec{r}\frac{d^4\vec{r}}{dt^4}] + 3[\ddot{\vec{r}}\ddot{\vec{r}}])\vec{r}}{r^9} \\
&+ \frac{22050([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])^2(4[\dot{\vec{r}}\frac{d^3\vec{r}}{dt^3}] + [\vec{r}\frac{d^4\vec{r}}{dt^4}] + 3[\ddot{\vec{r}}\ddot{\vec{r}}])\vec{r}}{r^9} - \frac{6300([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])(4[\dot{\vec{r}}\frac{d^3\vec{r}}{dt^3}] + [\vec{r}\frac{d^4\vec{r}}{dt^4}] + 3[\ddot{\vec{r}}\ddot{\vec{r}}])\ddot{\vec{r}}}{r^7} \\
&+ \frac{88200[\vec{r}\dot{\vec{r}}]([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])(4[\dot{\vec{r}}\frac{d^3\vec{r}}{dt^3}] + [\vec{r}\frac{d^4\vec{r}}{dt^4}] + 3[\ddot{\vec{r}}\ddot{\vec{r}}])\dot{\vec{r}}}{r^9} \\
&- \frac{396900[\vec{r}\dot{\vec{r}}]^2([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])(4[\dot{\vec{r}}\frac{d^3\vec{r}}{dt^3}] + [\vec{r}\frac{d^4\vec{r}}{dt^4}] + 3[\ddot{\vec{r}}\ddot{\vec{r}}])\vec{r}}{r^{11}} \\
&+ \frac{210(4[\dot{\vec{r}}\frac{d^3\vec{r}}{dt^3}] + [\vec{r}\frac{d^4\vec{r}}{dt^4}] + 3[\ddot{\vec{r}}\ddot{\vec{r}}])\frac{d^4\vec{r}}{dt^4}}{r^5} - \frac{4200[\vec{r}\dot{\vec{r}}](4[\dot{\vec{r}}\frac{d^3\vec{r}}{dt^3}] + [\vec{r}\frac{d^4\vec{r}}{dt^4}] + 3[\ddot{\vec{r}}\ddot{\vec{r}}])\frac{d^3\vec{r}}{dt^3}}{r^7} \\
&+ \frac{44100[\vec{r}\dot{\vec{r}}]^2(4[\dot{\vec{r}}\frac{d^3\vec{r}}{dt^3}] + [\vec{r}\frac{d^4\vec{r}}{dt^4}] + 3[\ddot{\vec{r}}\ddot{\vec{r}}])\ddot{\vec{r}}}{r^9} - \frac{264600[\vec{r}\dot{\vec{r}}]^3(4[\dot{\vec{r}}\frac{d^3\vec{r}}{dt^3}] + [\vec{r}\frac{d^4\vec{r}}{dt^4}] + 3[\ddot{\vec{r}}\ddot{\vec{r}}])\dot{\vec{r}}}{r^{11}} \\
&+ \frac{727650[\vec{r}\dot{\vec{r}}]^4(4[\dot{\vec{r}}\frac{d^3\vec{r}}{dt^3}] + [\vec{r}\frac{d^4\vec{r}}{dt^4}] + 3[\ddot{\vec{r}}\ddot{\vec{r}}])\vec{r}}{r^{13}} + \frac{29400([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])((3[\dot{\vec{r}}\ddot{\vec{r}}] + [\vec{r}\frac{d^3\vec{r}}{dt^3}])^2)\vec{r}}{r^9} \\
&- \frac{4200(3[\dot{\vec{r}}\ddot{\vec{r}}] + [\vec{r}\frac{d^3\vec{r}}{dt^3}])^2\ddot{\vec{r}}}{r^7} + \frac{58800[\vec{r}\dot{\vec{r}}](3 * [\dot{\vec{r}}\ddot{\vec{r}}] + [\vec{r}\frac{d^3\vec{r}}{dt^3}])^2\dot{\vec{r}}}{r^9} - \frac{264600[\vec{r}\dot{\vec{r}}]^2(3[\dot{\vec{r}}\ddot{\vec{r}}] + [\vec{r}\frac{d^3\vec{r}}{dt^3}])^2\vec{r}}{r^{11}} \\
&+ \frac{88200([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])^2(3[\dot{\vec{r}}\ddot{\vec{r}}] + [\vec{r}\frac{d^3\vec{r}}{dt^3}])\dot{\vec{r}}}{r^9} - \frac{793800[\vec{r}\dot{\vec{r}}]([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])^2(3[\dot{\vec{r}}\ddot{\vec{r}}] + [\vec{r}\frac{d^3\vec{r}}{dt^3}])\vec{r}}{r^{11}}
\end{aligned}
$$

$$
\begin{aligned}
&- \frac{8400([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])((3[\dot{\vec{r}}\ddot{\vec{r}}]) + [\vec{r}\frac{d^3\vec{r}}{dt^3}])\frac{d^3\vec{r}}{dt^3}}{r^7} + \frac{176400[\vec{r}\dot{\vec{r}}]([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])(3[\dot{\vec{r}}\ddot{\vec{r}}] + [\vec{r}\frac{d^3\vec{r}}{dt^3}])\ddot{\vec{r}}}{r^9} \\
&- \frac{1587600[\vec{r}\dot{\vec{r}}]^2([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])(3[\dot{\vec{r}}\ddot{\vec{r}}] + [\vec{r}\frac{d^3\vec{r}}{dt^3}])\dot{\vec{r}}}{r^{11}} + \frac{5821200[\vec{r}\dot{\vec{r}}]^3([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])(3[\dot{\vec{r}}\ddot{\vec{r}}] + [\vec{r}\frac{d^3\vec{r}}{dt^3}])\vec{r}}{r^{13}} \\
&+ \frac{168(3[\dot{\vec{r}}\ddot{\vec{r}}] + [\vec{r}\frac{d^3\vec{r}}{dt^3}])\frac{d^5\vec{r}}{dt^5}}{r^5} - \frac{4200[\vec{r}\dot{\vec{r}}](3[\dot{\vec{r}}\ddot{\vec{r}}] + [\vec{r}\frac{d^3\vec{r}}{dt^3}])\frac{d^4\vec{r}}{dt^4}}{r^7} + \frac{58800[\vec{r}\dot{\vec{r}}]^2((3[\dot{\vec{r}}\ddot{\vec{r}}]) + [\vec{r}\frac{d^3\vec{r}}{dt^3}])\frac{d^3\vec{r}}{dt^3}}{r^9} \\
&- \frac{529200[\vec{r}\dot{\vec{r}}]^3(3[\dot{\vec{r}}\ddot{\vec{r}}] + [\vec{r}\frac{d^3\vec{r}}{dt^3}])\ddot{\vec{r}}}{r^{11}} + \frac{2910600[\vec{r}\dot{\vec{r}}]^4(3[\dot{\vec{r}}\ddot{\vec{r}}] + [\vec{r}\frac{d^3\vec{r}}{dt^3}])\dot{\vec{r}}}{r^{13}} - \frac{7567560[\vec{r}\dot{\vec{r}}]^5(3[\dot{\vec{r}}\ddot{\vec{r}}] + [\vec{r}\frac{d^3\vec{r}}{dt^3}])\vec{r}}{r^{15}} \\
&- \frac{99225([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])^4\vec{r}}{r^{11}} + \frac{44100([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])^3\ddot{\vec{r}}}{r^9} - \frac{793800[\vec{r}\dot{\vec{r}}]([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])^3\dot{\vec{r}}}{r^{11}} \\
&+ \frac{4365900[\vec{r}\dot{\vec{r}}]^2([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])^3\vec{r}}{r^{13}} - \frac{3150([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])^2\frac{d^4\vec{r}}{dt^4}}{r^7} + \frac{88200[\vec{r}\dot{\vec{r}}]([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])^2\frac{d^3\vec{r}}{dt^3}}{r^9} \\
&- \frac{1190700[\vec{r}\dot{\vec{r}}]^2([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])^2\ddot{\vec{r}}}{r^{11}} + \frac{8731800[\vec{r}\dot{\vec{r}}]^3([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])^2\dot{\vec{r}}}{r^{13}} - \frac{28378350[\vec{r}\dot{\vec{r}}]^4([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])^2\vec{r}}{r^{15}} \\
&+ \frac{84([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])\frac{d^6\vec{r}}{dt^6}}{r^5} - \frac{2520[\vec{r}\dot{\vec{r}}]([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])\frac{d^5\vec{r}}{dt^5}}{r^7} + \frac{44100[\vec{r}\dot{\vec{r}}]^2([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])\frac{d^4\vec{r}}{dt^4}}{r^9} \\
&- \frac{529200[\vec{r}\dot{\vec{r}}]^3([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])\frac{d^3\vec{r}}{dt^3}}{r^{11}} + \frac{4365900[\vec{r}\dot{\vec{r}}]^4([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])\ddot{\vec{r}}}{r^{13}} - \frac{22702680[\vec{r}\dot{\vec{r}}]^5([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])\dot{\vec{r}}}{r^{15}} \\
&+ \frac{56756700[\vec{r}\dot{\vec{r}}]^6([\vec{r}\ddot{\vec{r}}] + [\dot{\vec{r}}\dot{\vec{r}}])\vec{r}}{r^{17}} - \frac{\frac{d^8\vec{r}}{dt^8}}{r^3} + \frac{24[\vec{r}\dot{\vec{r}}]\frac{d^7\vec{r}}{dt^7}}{r^5} - \frac{420[\vec{r}\dot{\vec{r}}]^2\frac{d^6\vec{r}}{dt^6}}{r^7} + \frac{5880[\vec{r}\dot{\vec{r}}]^3\frac{d^5\vec{r}}{dt^5}}{r^9} - \frac{66150[\vec{r}\dot{\vec{r}}]^4\frac{d^4\vec{r}}{dt^4}}{r^{11}} \\
&+ \frac{582120[\vec{r}\dot{\vec{r}}]^5\frac{d^3\vec{r}}{dt^3}}{r^{13}} - \frac{3783780[\vec{r}\dot{\vec{r}}]^6\ddot{\vec{r}}}{r^{15}} + \frac{16216200[\vec{r}\dot{\vec{r}}]^7\dot{\vec{r}}}{r^{17}} - \frac{34459425[\vec{r}\dot{\vec{r}}]^8\vec{r}}{r^{19}}]
\end{aligned}
$$

Die folgenden Ableitungen werden ausschließlich durch $\vec{r}$ und $\dot{\vec{r}}$ ausgedrückt. Dies wurde durch Rücksubstitution erreicht. Diese wird in Abschnitt B genauer betrachtet. Dazu sind insbesondere folgende Ableitungen hilfreich:

$\mathbf{f(t)}$	$\mathbf{\frac{df(t)}{dt}}$
$[\vec{r}\vec{r}]$	$2[\vec{r}\dot{\vec{r}}]$
$[\vec{r}\dot{\vec{r}}]$	$[\vec{r}\ddot{\vec{r}}][\dot{\vec{r}}\dot{\vec{r}}]$
$[\dot{\vec{r}}\dot{\vec{r}}]$	$2[\dot{\vec{r}}\ddot{\vec{r}}]$
$\frac{1}{r^p}$	$-\frac{p[\vec{r}\dot{\vec{r}}]}{r^{p+1}}$
$\frac{n[\vec{r}\vec{r}]^m}{r^p}$	$\frac{2n(m-\frac{p}{2})[\vec{r}\dot{\vec{r}}]}{r^{p-m+1}}$
$\frac{n[\vec{r}\dot{\vec{r}}]^m}{r^p}$	$\frac{nm[\vec{r}\dot{\vec{r}}]^{m-1}[\vec{r}\ddot{\vec{r}}]}{r^p} - \frac{np[\vec{r}\dot{\vec{r}}]^{m+1}}{r^{p+1}}$
$\frac{n[\dot{\vec{r}}\dot{\vec{r}}]^m}{r^p}$	$\frac{nm[\dot{\vec{r}}\dot{\vec{r}}]^{m-1}[\dot{\vec{r}}\ddot{\vec{r}}]}{r^p} - \frac{np[\vec{r}\dot{\vec{r}}][\dot{\vec{r}}\dot{\vec{r}}]^m}{r^{p+1}}$

C.0.2 3.Ableitung nach der Zeit

$$\frac{d^3\vec{r}}{dt^3} = \mu(\frac{3[\vec{r}\dot{\vec{r}}]}{r^5}\vec{r} - \frac{1}{r^3}\dot{\vec{r}})$$

mit

$$h_3^0 = \frac{3[\vec{r}\dot{\vec{r}}]}{r^5} = \frac{dh_3^1}{dt} = \frac{dh_2^0}{dt}$$
$$h_3^1 = -\frac{1}{r^3} = h_2^0$$

Wobei h_3^0 ein skalarer Faktor von $\vec{r}$ und h_3^1 ein skalarer Faktor von $\dot{\vec{r}}$ ist.
Dadurch lässt sich $\frac{d^3\vec{r}}{dt^3}$ wie folgt schreiben:

$$\frac{d^3\vec{r}}{dt^3} = \mu\langle h_3^0\vec{r} + h_3^1\dot{\vec{r}}\rangle$$
$$\frac{d^3\vec{r}}{dt^3} = \mu\langle(\frac{dh_3^1}{dt})\vec{r} + h_3^1\dot{\vec{r}}\rangle$$
$$\frac{d^3\vec{r}}{dt^3} = \mu\langle(\frac{dh_2^0}{dt})\vec{r} + h_2^0\dot{\vec{r}}\rangle$$

C.0.3 4.Ableitung nach der Zeit

$$\frac{d^4\vec{r}}{dt^4} = \frac{\mu}{r^8}\langle(-15r[\vec{r}\dot{\vec{r}}]^2 - 3\mu[\vec{r}\vec{r}] + \mu r^2 + 3r^3[\dot{\vec{r}}\dot{\vec{r}}])\vec{r} + (6r^3[\vec{r}\dot{\vec{r}}])\dot{\vec{r}}\rangle$$
$$= \mu\langle(\frac{3[\dot{\vec{r}}\dot{\vec{r}}]}{r^5} + \frac{\mu}{r^6} - \frac{15[\vec{r}\dot{\vec{r}}]^2}{r^7} - \frac{3\mu[\vec{r}\vec{r}]}{r^8})\vec{r} + (\frac{6[\vec{r}\dot{\vec{r}}]}{r^5})\dot{\vec{r}}\rangle$$

mit

$$h_4^0 = \frac{3[\dot{\vec{r}}\dot{\vec{r}}]}{r^5} + \frac{\mu}{r^6} - \frac{15[\vec{r}\dot{\vec{r}}]^2}{r^7} - \frac{3\mu[\vec{r}\vec{r}]}{r^8}$$
$$h_4^0 = \frac{1}{2}(\frac{6[\dot{\vec{r}}\dot{\vec{r}}]}{r^5} - \frac{30[\vec{r}\dot{\vec{r}}]^2}{r^7} - \frac{\mu[\vec{r}\vec{r}]}{r^8}) + \frac{\mu}{r^6}$$
$$h_4^0 = \frac{1}{2}(\frac{dh_3^0}{dt}) + \frac{\mu}{r^6}$$
$$h_4^1 = \frac{6[\vec{r}\dot{\vec{r}}]}{r^5} = 2(\frac{3[\vec{r}\dot{\vec{r}}]}{r^5}) = 2h_3^0$$

Wobei h_4^0 ein skalarer Faktor von $\vec{r}$ und h_4^1 ein skalarer Faktor von $\dot{\vec{r}}$ ist.
Dadurch lässt sich $\frac{d^4\vec{r}}{dt^4}$ wie folgt schreiben:

$$\frac{d^4\vec{r}}{dt^4} = \mu\langle h_4^0\vec{r} + h_4^1\dot{\vec{r}}\rangle$$
$$= \mu\langle(\frac{1}{2}(\frac{dh_3^0}{dt}) + \frac{\mu}{r^6})\vec{r} + 2h_3^0\dot{\vec{r}}\rangle$$

C.0.4 5.Ableitung nach der Zeit

$$\frac{d^5\vec{r}}{dt^5} = \mu[\frac{9[\dot{\vec{r}}\dot{\vec{r}}]\dot{\vec{r}}}{r^5} - \frac{45[\vec{r}\dot{\vec{r}}][\dot{\vec{r}}\dot{\vec{r}}]\vec{r}}{r^7} - \frac{45[\vec{r}\dot{\vec{r}}]^2\dot{\vec{r}}}{r^7} - \frac{9\mu[\vec{r}\vec{r}]\dot{\vec{r}}}{r^8} + \frac{\mu\dot{\vec{r}}}{r^6} + \frac{105[\vec{r}\dot{\vec{r}}]^3\vec{r}}{r^9} + \frac{54\mu[\vec{r}\vec{r}][\vec{r}\dot{\vec{r}}]\vec{r}}{r^{10}}$$
$$- \frac{24\mu[\vec{r}\dot{\vec{r}}]\vec{r}}{r^8}]$$

Indem $\frac{d^5\vec{r}}{dt^5}$ als Linearkombination aus $\vec{r}$ und $\dot{\vec{r}}$ dargestellt wird

$$\frac{d^5\vec{r}}{dt^5} = \mu\langle(-\frac{24\mu[\vec{r}\dot{\vec{r}}]}{r^8} - \frac{45[\vec{r}\dot{\vec{r}}][\dot{\vec{r}}\dot{\vec{r}}]}{r^7} + \frac{105[\vec{r}\dot{\vec{r}}]^3}{r^9} + \frac{54\mu[\vec{r}\vec{r}][\vec{r}\dot{\vec{r}}]}{r^{10}})\vec{r}$$
$$+ (\frac{9[\dot{\vec{r}}\dot{\vec{r}}]}{r^5} + \frac{\mu}{r^6} - \frac{45[\vec{r}\dot{\vec{r}}]^2}{r^7} - \frac{9\mu[\vec{r}\vec{r}]}{r^8})\dot{\vec{r}}\rangle$$

mit

$$h_5^0 = -\frac{24\mu[\vec{r}\dot{\vec{r}}]}{r^8} - \frac{45[\vec{r}\dot{\vec{r}}][\dot{\vec{r}}\dot{\vec{r}}]}{r^7} + \frac{105[\vec{r}\dot{\vec{r}}]^3}{r^9} + \frac{54\mu[\vec{r}\vec{r}][\vec{r}\dot{\vec{r}}]}{r^{10}}$$
$$h_5^1 = \frac{9[\dot{\vec{r}}\dot{\vec{r}}]}{r^5} + \frac{\mu}{r^6} - \frac{45[\vec{r}\dot{\vec{r}}]^2}{r^7} - \frac{9\mu[\vec{r}\vec{r}]}{r^8}$$
$$= 3(\frac{3[\dot{\vec{r}}\dot{\vec{r}}]}{r^5} + \frac{\mu}{r^6} - \frac{15[\vec{r}\dot{\vec{r}}]^2}{r^7} - \frac{3\mu[\vec{r}\vec{r}]}{r^8}) - \frac{2\mu}{r^6}$$
$$= 3(h_4^0) - \frac{2\mu}{r^6}$$

kann $\frac{d^5\vec{r}}{dt^5}$ wie folgt geschrieben werden:

$$\frac{d^5\vec{r}}{dt^5} = \mu\langle h_5^0\vec{r} + h_5^1\dot{\vec{r}}\rangle$$
$$\frac{d^5\vec{r}}{dt^5} = \mu\langle h_5^0\vec{r} + (3(h_4^0) - \frac{2\mu}{r^6})\dot{\vec{r}}\rangle$$

C.0.5 6.Ableitung nach der Zeit

Indem $\frac{d^6\vec{r}}{dt^6}$ als Linearkombination aus $\vec{r}$ und $\dot{\vec{r}}$ dargestellt wird

$$\frac{d^6\vec{r}}{dt^6} = \mu\langle(-\frac{45[\dot{\vec{r}}\dot{\vec{r}}]^2}{r^7} - \frac{33\mu[\dot{\vec{r}}\dot{\vec{r}}]}{r^8} - \frac{\mu^2}{r^9} + \frac{630[\vec{r}\dot{\vec{r}}]^2[\dot{\vec{r}}\dot{\vec{r}}]}{r^9} + \frac{99\mu[\vec{r}\vec{r}][\dot{\vec{r}}\dot{\vec{r}}]}{r^{10}} + \frac{435\mu[\vec{r}\dot{\vec{r}}]^2}{r^{10}} + \frac{33\mu^2[\vec{r}\vec{r}]}{r^{11}} - \frac{945[\vec{r}\dot{\vec{r}}]^4}{r^{11}}$$
$$- \frac{855\mu[\vec{r}\vec{r}][\vec{r}\dot{\vec{r}}]^2}{r^{12}} - \frac{54\mu^2[\vec{r}\vec{r}]^2}{r^{13}})\vec{r}$$
$$+ (-\frac{180[\vec{r}\dot{\vec{r}}][\dot{\vec{r}}\dot{\vec{r}}]}{r^7} - \frac{66\mu[\vec{r}\dot{\vec{r}}]}{r^8} + \frac{420[\vec{r}\dot{\vec{r}}]^3}{r^9} + \frac{216\mu[\vec{r}\vec{r}][\vec{r}\dot{\vec{r}}]}{r^{10}})\dot{\vec{r}}\rangle$$

mit

$$h_6^0 = -\frac{45[\dot{\vec{r}}\dot{\vec{r}}]^2}{r^7} - \frac{33\mu[\dot{\vec{r}}\dot{\vec{r}}]}{r^8} - \frac{\mu^2}{r^9} + \frac{630[\vec{r}\dot{\vec{r}}]^2[\dot{\vec{r}}\dot{\vec{r}}]}{r^9} + \frac{99\mu[\vec{r}\vec{r}][\dot{\vec{r}}\dot{\vec{r}}]}{r^{10}} + \frac{435\mu[\vec{r}\dot{\vec{r}}]^2}{r^{10}} + \frac{33\mu^2[\vec{r}\vec{r}]}{r^{11}} - \frac{945[\vec{r}\dot{\vec{r}}]^4}{r^{11}}$$
$$- \frac{855\mu[\vec{r}\vec{r}][\vec{r}\dot{\vec{r}}]^2}{r^{12}} - \frac{54\mu^2[\vec{r}\vec{r}]^2}{r^{13}}$$
$$h_6^1 = -\frac{180[\vec{r}\dot{\vec{r}}][\dot{\vec{r}}\dot{\vec{r}}]}{r^7} - \frac{66\mu[\vec{r}\dot{\vec{r}}]}{r^8} + \frac{420[\vec{r}\dot{\vec{r}}]^3}{r^9} + \frac{216\mu[\vec{r}\vec{r}][\vec{r}\dot{\vec{r}}]}{r^{10}}$$

kann $\frac{d^6\vec{r}}{dt^6}$ wie folgt geschrieben werden:

$$\frac{d^6\vec{r}}{dt^6} = \mu\langle h_6^0\vec{r} + h_6^1\dot{\vec{r}}\rangle$$
$$\frac{d^6\vec{r}}{dt^6} = \mu\langle h_6^0\vec{r} + (4h_5^0 + \frac{30\mu[\vec{r}\dot{\vec{r}}]}{r^8})\dot{\vec{r}}\rangle$$

C.0.6 7.Ableitung nach der Zeit

$$
\begin{aligned}
\frac{d^7\vec r}{dt^7} = \mu\langle(&\frac{1575[\vec r\dot{\vec r}][\dot{\vec r}\dot{\vec r}]^2}{r^9} + \frac{1692\mu[\vec r\dot{\vec r}][\dot{\vec r}\dot{\vec r}]}{r^{10}} + \frac{207\mu^2[\vec r\dot{\vec r}]}{r^{11}} - \frac{9450[\vec r\dot{\vec r}]^3[\dot{\vec r}\dot{\vec r}]}{r^{11}} - \frac{3960\mu[\vec r\vec r][\vec r\dot{\vec r}][\dot{\vec r}\dot{\vec r}]}{r^{12}} \\
&- \frac{7740\mu[\vec r\dot{\vec r}]^3}{r^{12}} - \frac{1863\mu^2[\vec r\vec r][\vec r\dot{\vec r}]}{r^{13}} + \frac{10395[\vec r\dot{\vec r}]^5}{r^{13}} + \frac{14040\mu[\vec r\vec r][\vec r\dot{\vec r}]^3}{r^{14}} + \frac{2412\mu^2[\vec r\vec r]^2[\vec r\dot{\vec r}]}{r^{15}})\vec r \\
&+ (-\frac{225[\dot{\vec r}\dot{\vec r}]^2}{r^7} - \frac{99\mu[\dot{\vec r}\dot{\vec r}]}{r^8} - \frac{\mu^2}{r^9} + \frac{3150[\vec r\dot{\vec r}]^2[\dot{\vec r}\dot{\vec r}]}{r^9} + \frac{495\mu[\vec r\vec r][\dot{\vec r}\dot{\vec r}]}{r^{10}} \\
&+ \frac{1755\mu[\vec r\dot{\vec r}]^2}{r^{10}} + \frac{99\mu^2[\vec r\vec r]}{r^{11}} - \frac{4725[\vec r\dot{\vec r}]^4}{r^{11}} - \frac{4275\mu[\vec r\vec r][\vec r\dot{\vec r}]^2}{r^{12}} - \frac{270\mu^2[\vec r\vec r]^2}{r^{13}})\dot{\vec r}\rangle
\end{aligned}
$$

Indem $\frac{d^7\vec r}{dt^7}$ als Linearkombination aus $\vec r$ und $\dot{\vec r}$ dargestellt wird

$$
\begin{aligned}
h_7^0 = &\frac{1575[\vec r\dot{\vec r}][\dot{\vec r}\dot{\vec r}]^2}{r^9} + \frac{1692\mu[\vec r\dot{\vec r}][\dot{\vec r}\dot{\vec r}]}{r^{10}} + \frac{207\mu^2[\vec r\dot{\vec r}]}{r^{11}} - \frac{9450[\vec r\dot{\vec r}]^3[\dot{\vec r}\dot{\vec r}]}{r^{11}} - \frac{3960\mu[\vec r\vec r][\vec r\dot{\vec r}][\dot{\vec r}\dot{\vec r}]}{r^{12}} \\
&- \frac{7740\mu[\vec r\dot{\vec r}]^3}{r^{12}} - \frac{1863\mu^2[\vec r\vec r][\vec r\dot{\vec r}]}{r^{13}} + \frac{10395[\vec r\dot{\vec r}]^5}{r^{13}} + \frac{14040\mu[\vec r\vec r][\vec r\dot{\vec r}]^3}{r^{14}} + \frac{2412\mu^2[\vec r\vec r]^2[\vec r\dot{\vec r}]}{r^{15}} \\
h_7^1 = &-\frac{225[\dot{\vec r}\dot{\vec r}]^2}{r^7} - \frac{99\mu[\dot{\vec r}\dot{\vec r}]}{r^8} - \frac{\mu^2}{r^9} + \frac{3150[\vec r\dot{\vec r}]^2[\dot{\vec r}\dot{\vec r}]}{r^9} + \frac{495\mu[\vec r\vec r][\dot{\vec r}\dot{\vec r}]}{r^{10}} \\
&+ \frac{1755\mu[\vec r\dot{\vec r}]^2}{r^{10}} + \frac{99\mu^2[\vec r\vec r]}{r^{11}} - \frac{4725[\vec r\dot{\vec r}]^4}{r^{11}} - \frac{4275\mu[\vec r\vec r][\vec r\dot{\vec r}]^2}{r^{12}} - \frac{270\mu^2[\vec r\vec r]^2}{r^{13}}
\end{aligned}
$$

kann $\frac{d^7\vec r}{dt^7}$ wie folgt geschrieben werden:

$$
\begin{aligned}
\frac{d^7\vec r}{dt^7} &= \mu\langle h_7^0\vec r + h_7^1\dot{\vec r}\rangle \\
\frac{d^7\vec r}{dt^7} &= \mu\langle h_7^0\vec r + (5h_6^0 + (\frac{66\mu[\dot{\vec r}\dot{\vec r}]}{r^8} + \frac{4\mu^2}{r^9} - \frac{420\mu[\dot{\vec r}\dot{\vec r}]}{r^{10}} - \frac{66\mu^2[\vec r\vec r]}{r^{11}})\dot{\vec r})\rangle
\end{aligned}
$$

C.0.7 8.Ableitung nach der Zeit

$$
\begin{aligned}
\frac{d^8\vec r}{dt^8} = \mu\langle(&\frac{1575[\dot{\vec r}\dot{\vec r}]^3}{r^9} + \frac{1917\mu[\dot{\vec r}\dot{\vec r}]^2}{r^{10}} + \frac{306\mu^2[\dot{\vec r}\dot{\vec r}]}{r^{11}} - \frac{42525[\vec r\dot{\vec r}]^2[\dot{\vec r}\dot{\vec r}]^2}{r^{11}} + \frac{\mu^3}{r^{12}} - \frac{5535\mu[\vec r\vec r][\dot{\vec r}\dot{\vec r}]^2}{r^{12}} \\
&- \frac{57510\mu[\vec r\dot{\vec r}]^2[\dot{\vec r}\dot{\vec r}]}{r^{12}} - \frac{4050\mu^2[\vec r\vec r][\dot{\vec r}\dot{\vec r}]}{r^{13}} - \frac{11142\mu^2[\vec r\dot{\vec r}]^2}{r^{13}} + \frac{155925[\vec r\dot{\vec r}]^4[\dot{\vec r}\dot{\vec r}]}{r^{13}} - \frac{306\mu^3[\vec r\vec r]}{r^{14}} \\
&+ \frac{117990\mu[\vec r\vec r][\vec r\dot{\vec r}]^2[\dot{\vec r}\dot{\vec r}]}{r^{14}} + \frac{144585\mu[\vec r\dot{\vec r}]^4}{r^{14}} + \frac{6372\mu[\vec r\vec r]^2[\dot{\vec r}\dot{\vec r}]}{r^{15}} + \frac{69282\mu[\vec r\vec r][\vec r\dot{\vec r}]^2}{r^{15}} \\
&- \frac{135135[\vec r\dot{\vec r}]^6}{r^{15}} + \frac{2133\mu^3[\vec r\vec r]^2}{r^{16}} - \frac{248535\mu[\vec r\vec r][\vec r\dot{\vec r}]^4}{r^{16}} - \frac{78300\mu[\vec r\vec r]^2[\vec r\dot{\vec r}]^2}{r^{17}} - \frac{2412\mu^3[\vec r\vec r]^3}{r^{18}})\vec r \\
&+ (\frac{9450[\vec r\dot{\vec r}][\dot{\vec r}\dot{\vec r}]^2}{r^9} + \frac{7884\mu[\vec r\dot{\vec r}][\dot{\vec r}\dot{\vec r}]}{r^{10}} + \frac{612\mu^2[\vec r\dot{\vec r}]}{r^{11}} - \frac{56700[\vec r\dot{\vec r}]^3[\dot{\vec r}\dot{\vec r}]}{r^{11}} - \frac{23760\mu[\vec r\vec r][\vec r\dot{\vec r}][\dot{\vec r}\dot{\vec r}]}{r^{12}} \\
&- \frac{40140\mu[\vec r\dot{\vec r}]^3}{r^{12}} - \frac{8532\mu^2[\vec r\vec r][\vec r\dot{\vec r}]}{r^13} + \frac{62370[\vec r\dot{\vec r}]^5}{r^{13}} + \frac{84240\mu[\vec r\vec r][\vec r\dot{\vec r}]^3}{r^{14}} + \frac{14472\mu[\vec r\vec r]^2[\vec r\dot{\vec r}]}{r^{15}})\dot{\vec r}\rangle
\end{aligned}
$$

Indem $\frac{d^8\vec r}{dt^8}$ als Linearkombination aus $\vec r$ und $\dot{\vec r}$ dargestellt wird

$$
\begin{aligned}
h_8^0 = {} & \frac{1575[\dot{\vec r}\dot{\vec r}]^3}{r^9} + \frac{1917\mu[\dot{\vec r}\dot{\vec r}]^2}{r^{10}} + \frac{306\mu^2[\dot{\vec r}\dot{\vec r}]}{r^{11}} - \frac{42525[\vec r\dot{\vec r}]^2[\dot{\vec r}\dot{\vec r}]^2}{r^{11}} + \frac{\mu^3}{r^{12}} - \frac{5535\mu[\vec r\vec r][\dot{\vec r}\dot{\vec r}]^2}{r^{12}} \\
& - \frac{57510\mu[\vec r\dot{\vec r}]^2[\dot{\vec r}\dot{\vec r}]}{r^{12}} - \frac{4050\mu^2[\vec r\vec r][\dot{\vec r}\dot{\vec r}]}{r^{13}} - \frac{11142\mu^2[\vec r\dot{\vec r}]^2}{r^{13}} + \frac{155925[\vec r\dot{\vec r}]^4[\dot{\vec r}\dot{\vec r}]}{r^{13}} - \frac{306\mu^3[\vec r\vec r]}{r^{14}} \\
& + \frac{117990\mu[\vec r\vec r][\vec r\dot{\vec r}]^2[\dot{\vec r}\dot{\vec r}]}{r^{14}} + \frac{144585\mu[\vec r\dot{\vec r}]^4}{r^{14}} + \frac{6372\mu[\vec r\vec r]^2[\dot{\vec r}\dot{\vec r}]}{r^{15}} + \frac{69282\mu[\vec r\vec r][\vec r\dot{\vec r}]^2}{r^{15}} \\
& - \frac{135135[\vec r\dot{\vec r}]^6}{r^{15}} + \frac{2133\mu^3[\vec r\vec r]^2}{r^{16}} - \frac{248535\mu[\vec r\vec r][\vec r\dot{\vec r}]^4}{r^{16}} - \frac{78300\mu[\vec r\vec r]^2[\vec r\dot{\vec r}]^2}{r^{17}} - \frac{2412\mu^3[\vec r\vec r]^3}{r^{18}} \\
h_8^1 = {} & \frac{9450[\vec r\dot{\vec r}][\dot{\vec r}\dot{\vec r}]^2}{r^9} + \frac{7884\mu[\vec r\dot{\vec r}][\dot{\vec r}\dot{\vec r}]}{r^{10}} + \frac{612\mu^2[\vec r\dot{\vec r}]}{r^{11}} - \frac{56700[\vec r\dot{\vec r}]^3[\dot{\vec r}\dot{\vec r}]}{r^{11}} - \frac{23760\mu[\vec r\vec r][\vec r\dot{\vec r}][\dot{\vec r}\dot{\vec r}]}{r^{12}} \\
& - \frac{40140\mu[\vec r\dot{\vec r}]^3}{r^{12}} - \frac{8532\mu^2[\vec r\vec r][\vec r\dot{\vec r}]}{r^{13}} + \frac{62370[\vec r\dot{\vec r}]^5}{r^{13}} + \frac{84240\mu[\vec r\vec r][\vec r\dot{\vec r}]^3}{r^{14}} + \frac{14472\mu[\vec r\vec r]^2[\vec r\dot{\vec r}]}{r^{15}}
\end{aligned}
$$

kann $\frac{d^8\vec r}{dt^8}$ wie folgt geschrieben werden:

$$\frac{d^8\vec r}{dt^8} = \mu\langle h_8^0\vec r + h_8^1\dot{\vec r}\rangle$$

C.0.8 9.Ableitung nach der Zeit

$$
\begin{aligned}
\frac{d^9\vec r}{dt^9} = \mu\langle & -\frac{99225[\vec r\dot{\vec r}][\dot{\vec r}\dot{\vec r}]^3\vec r}{r^{11}} - \frac{164160\mu[\vec r\dot{\vec r}][\dot{\vec r}\dot{\vec r}]^2\vec r}{r^{12}} - \frac{49302\mu^2[\vec r\dot{\vec r}][\dot{\vec r}\dot{\vec r}]\vec r}{r^{13}} \\
& + \frac{1091475[\vec r\dot{\vec r}]^3[\dot{\vec r}\dot{\vec r}]^2\vec r}{r^{13}} - \frac{1848\mu^3[\vec r\dot{\vec r}]\vec r}{r^{14}} + \frac{387450\mu[\vec r\vec r][\vec r\dot{\vec r}][\dot{\vec r}\dot{\vec r}]^2\vec r}{r^{14}} \\
& + \frac{1731240\mu[\vec r\dot{\vec r}]^3[\dot{\vec r}\dot{\vec r}]\vec r}{r^{14}} + \frac{377622\mu^2[\vec r\vec r][\vec r\dot{\vec r}][\dot{\vec r}\dot{\vec r}]\vec r}{r^{15}} + \frac{438570\mu^2[\vec r\dot{\vec r}]^3\vec r}{r^{15}} \\
& - \frac{2837835[\vec r\dot{\vec r}]^5[\dot{\vec r}\dot{\vec r}]\vec r}{r^{15}} + \frac{51732\mu^3[\vec r\vec r][\vec r\dot{\vec r}]\vec r}{r^{16}} - \frac{2895480\mu[\vec r\dot{\vec r}]^5\vec r}{r^{16}} \\
& - \frac{3269700\mu[\vec r\vec r][\vec r\dot{\vec r}]^3[\dot{\vec r}\dot{\vec r}]\vec r}{r^{16}} - \frac{488160\mu^2[\vec r\vec r]^2[\vec r\dot{\vec r}][\dot{\vec r}\dot{\vec r}]\vec r}{r^{17}} - \frac{2250990\mu^2[\vec r\vec r][\vec r\dot{\vec r}]^3\vec r}{r^{17}} \\
& + \frac{2027025[\vec r\dot{\vec r}]^7\vec r}{r^{17}} - \frac{214380\mu^3[\vec r\vec r]^2[\vec r\dot{\vec r}]\vec r}{r^{18}} + \frac{4787370\mu[\vec r\vec r][\vec r\dot{\vec r}]^5\vec r}{r^{18}} \\
& + \frac{2325240\mu^2[\vec r\vec r]^2[\vec r\dot{\vec r}]^3\vec r}{r^{19}} + \frac{200016\mu^3[\vec r\vec r]^3[\vec r\dot{\vec r}]\vec r}{r^{20}} + \frac{11025[\dot{\vec r}\dot{\vec r}]^3\dot{\vec r}}{r^9} \\
& + \frac{9801\mu[\dot{\vec r}\dot{\vec r}]^2\dot{\vec r}}{r^{10}} + \frac{918\mu^2[\dot{\vec r}\dot{\vec r}]\dot{\vec r}}{r^{11}} - \frac{297675[\vec r\dot{\vec r}]^2[\dot{\vec r}\dot{\vec r}]^2\dot{\vec r}}{r^{11}} + \frac{\mu^3\dot{\vec r}}{r^{12}} - \frac{38745\mu[\vec r\vec r][\dot{\vec r}\dot{\vec r}]^2\dot{\vec r}}{r^{12}} \\
& - \frac{342090\mu[\vec r\dot{\vec r}]^2[\dot{\vec r}\dot{\vec r}]\dot{\vec r}}{r^{12}} - \frac{20466\mu^2[\vec r\vec r][\dot{\vec r}\dot{\vec r}]\dot{\vec r}}{r^{13}} - \frac{50706\mu^2[\vec r\dot{\vec r}]^2\dot{\vec r}}{r^{13}} + \frac{1091475[\vec r\dot{\vec r}]^4[\dot{\vec r}\dot{\vec r}]\dot{\vec r}}{r^{13}} \\
& - \frac{918\mu^3[\vec r\vec r]\dot{\vec r}}{r^{14}} + \frac{908145\mu[\vec r\dot{\vec r}]^4\dot{\vec r}}{r^{14}} + \frac{825930\mu[\vec r\vec r][\vec r\dot{\vec r}]^2[\dot{\vec r}\dot{\vec r}]\dot{\vec r}}{r^{14}} \\
& + \frac{44604\mu^2[\vec r\vec r]^2[\dot{\vec r}\dot{\vec r}]\dot{\vec r}}{r^{15}} + \frac{406026\mu^2[\vec r\vec r][\vec r\dot{\vec r}]^2\dot{\vec r}}{r^{15}} - \frac{945945[\vec r\dot{\vec r}]^6\dot{\vec r}}{r^{15}} + \frac{10665\mu^3[\vec r\vec r]^2\dot{\vec r}}{r^{16}} \\
& - \frac{1739745\mu[\vec r\vec r][\vec r\dot{\vec r}]^4\dot{\vec r}}{r^{16}} - \frac{548100\mu^2[\vec r\vec r]^2[\vec r\dot{\vec r}]^2\dot{\vec r}}{r^{17}} - \frac{16884\mu^3[\vec r\vec r]^3\dot{\vec r}}{r^{18}}\rangle
\end{aligned}
$$

Anhang D

Höhere Ableitungen des Hauptproblems

Die Bewegungsgleichung des Hauptproblems ist wie folgt definiert:

$$\frac{d^2\vec{r}}{dt^2} = -\frac{\mu\vec{r}}{r^3} + \alpha \begin{pmatrix} \frac{x}{r^5} & - & \frac{5xz^2}{r^7} \\ \frac{y}{r^5} & - & \frac{5yz^2}{r^7} \\ \frac{3z}{r^5} & - & \frac{5z^3}{r^7} \end{pmatrix}$$

Dabei ist:

- $a_{\otimes}$ die große Halbachse der Erde.
- μ die Gravitationskonstante G multipliziert mit der Masse der Erde $M_{\otimes}$.
- C_{20} der zweite zonale Kugelfunktionskoeffizient des Gravitationspotential der Erde.
- t die Zeit.
- $\alpha = \frac{3}{2}\mu C_{20} a_{\otimes}^2$

Im Folgenden sind die höheren Ableitungen der Hauptproblems aufgeführt. Diese sind vollständig rücksubstituiert, d.h. jede der Ableitungen hängt ausschließlich vom Orts - u. Geschwindigkeitsvektor ab. Folglich sind keine Terme höherer Ableitungen enthalten.

$$\frac{d^0\vec{r}}{dt^0} = \vec{r}$$

$$\frac{d^1\vec{r}}{dt^1} = \dot{\vec{r}}$$

$$\frac{d^2\vec{r}}{dt^2} = -\mu\frac{\vec{r}}{r^3} + \alpha \begin{pmatrix} \frac{x}{r^5} & - & \frac{5xz^2}{r^7} \\ \frac{y}{r^5} & - & \frac{5yz^2}{r^7} \\ \frac{3z}{r^5} & - & \frac{5z^3}{r^7} \end{pmatrix}$$

$$\frac{d^3\vec{r}}{dt^3} = -\mu(\frac{\dot{\vec{r}}}{r^3} - \frac{3\vec{r}[\vec{r}\dot{\vec{r}}]}{r^5}) + \alpha \begin{pmatrix} \frac{\dot{x}}{r^5} & - & \frac{5x[\vec{r}\dot{\vec{r}}]+10xz\dot{z}+5z^2\dot{x}}{r^7} & + & \frac{35xz^2[\vec{r}\dot{\vec{r}}]}{r^9} \\ \frac{\dot{y}}{r^5} & - & \frac{5y[\vec{r}\dot{\vec{r}}]+10yz\dot{z}+5z^2\dot{y}}{r^7} & + & \frac{35yz^2[\vec{r}\dot{\vec{r}}]}{r^9} \\ \frac{3\dot{z}}{r^5} & - & \frac{15z[\vec{r}\dot{\vec{r}}]+10z^2\dot{z}}{r^7} & + & \frac{35z^3[\vec{r}\dot{\vec{r}}]}{r^9} \end{pmatrix}$$

Aus Platzgründen wird im Weiteren der Anteil des Keplerproblems zu $f_{Kep,i}$ zusammengefasst. Die ausführliche Darstellung ist in Kapitel C zu finden. In der abgekürzten Schreibweise steht i für den jeweiligen Grad der Ableitung. So wird beispielsweise die 1.Ableitung wie folgt bezeichnet:

$$f_{Kep,1} = \frac{d^1\vec{r}}{dt^1} = \dot{\vec{r}} = \begin{pmatrix} \dot{x}_{Kep,1} \\ \dot{y}_{Kep,1} \\ \dot{z}_{Kep,1} \end{pmatrix}$$

$$\begin{aligned}
\frac{d^4x}{dt^4} = x_{Kep,4} + \alpha\{ & \\
& x(-\frac{10\dot{z}^2 + 5[\dot{\vec{r}}\dot{\vec{r}}]}{r^7} - \frac{2\mu}{r^8} + \frac{35[\dot{\vec{r}}\dot{\vec{r}}]z^2 + 35[\vec{r}\dot{\vec{r}}]^2 + 140[\vec{r}\dot{\vec{r}}]z\dot{z}}{r^9} \\
& - \frac{26\mu(x^2+y^2) - 34\mu[\vec{r}\vec{r}] - \alpha}{r^{10}} - \frac{315[\vec{r}\dot{\vec{r}}]^2 z^2}{r^{11}} \\
& + \frac{(50\mu[\vec{r}\vec{r}] + 50\alpha)(x^2+y^2) - 50\mu[\vec{r}\vec{r}]^2 - 55\alpha[\vec{r}\vec{r}]}{r^{12}} \\
& + \frac{145\alpha z^4 + 60\alpha[\vec{r}\vec{r}]z^2}{r^{14}} - \frac{175\alpha[\vec{r}\vec{r}]z^4}{r^{16}}) \\
& + \dot{x}(-\frac{20z\dot{z} + 10[\vec{r}\dot{\vec{r}}]}{r^7} + \frac{70[\vec{r}\dot{\vec{r}}]z^2}{r^9})\}
\end{aligned}$$

$$\begin{aligned}
\frac{d^4y}{dt^4} = y_{Kep,4} + \alpha\{ & \\
& y(-\frac{10\dot{z}^2 + 5[\dot{\vec{r}}\dot{\vec{r}}]}{r^7} - \frac{2\mu}{r^8} + \frac{35[\vec{r}\dot{\vec{r}}]^2 + 35[\dot{\vec{r}}\dot{\vec{r}}]z^2 + 140[\vec{r}\dot{\vec{r}}]z\dot{z}}{r^9} \\
& - \frac{26\mu(x^2+y^2) - 34\mu[\vec{r}\vec{r}] - \alpha}{r^{10}} - \frac{315[\vec{r}\dot{\vec{r}}]^2 z^2}{r^{11}} \\
& + \frac{(50\mu[\vec{r}\vec{r}] + 50\alpha)(x^2+y^2) - 50\mu[\vec{r}\vec{r}]^2 - 55\alpha[\vec{r}\vec{r}]}{r^{12}} \\
& + \frac{145\alpha z^4 + 60\alpha[\vec{r}\vec{r}]z^2}{r^{14}} - \frac{175\alpha[\vec{r}\vec{r}]z^4}{r^{16}}) \\
& + \dot{y}(-\frac{20z\dot{z} + 10[\vec{r}\dot{\vec{r}}]}{r^7} + \frac{70[\vec{r}\dot{\vec{r}}]z^2}{r^9}\}
\end{aligned}$$

$$\begin{aligned}
\frac{d^4z}{dt^4} = z_{Kep,4} + \alpha\{ & \\
& z(-\frac{15[\dot{\vec{r}}\dot{\vec{r}}]}{r^7} - \frac{6\mu}{r^8} - \frac{35[\dot{\vec{r}}\dot{\vec{r}}](x^2+y^2) - 35[\vec{r}\vec{r}][\dot{\vec{r}}\dot{\vec{r}}] - 105[\vec{r}\dot{\vec{r}}]^2}{r^9} \\
& - \frac{26\mu(x^2+y^2) - 9\alpha - 44\mu[\vec{r}\vec{r}]}{r^{10}} + \frac{315[\vec{r}\dot{\vec{r}}]^2(x^2+y^2) - 315[\vec{r}\vec{r}][\vec{r}\dot{\vec{r}}]^2}{r^{11}} \\
& + \frac{(50\mu[\vec{r}\vec{r}] + 90\alpha)(x^2+y^2) - 50\mu[\vec{r}\vec{r}]^2 - 105\alpha[\vec{r}\vec{r}]}{r^{12}} \\
& + \frac{5\alpha(x^2+y^2-[\vec{r}\vec{r}])(29y^2+29x^2-51[\vec{r}\vec{r}])}{r^{14}} - \frac{175\alpha[\vec{r}\vec{r}]((x^2+y^2)-[\vec{r}\vec{r}])^2}{r^{16}}) \\
& + \dot{z}(-\frac{30([\vec{r}\dot{\vec{r}}] + z\dot{z})}{r^7} + \frac{210[\vec{r}\dot{\vec{r}}]z^2}{r^9})\}
\end{aligned}$$

$$
\begin{aligned}
\frac{d^5x}{dt^5} &= x_{Kep,5} + \alpha\{ \\
&x(\frac{210[\vec{r}\dot{\vec{r}}]\dot{z}^2 + 210[\dot{\vec{r}}\dot{\vec{r}}]z\dot{z} + 105[\vec{r}\dot{\vec{r}}][\dot{\vec{r}}\dot{\vec{r}}]}{r^9} + \frac{136\mu z\dot{z} + 32\mu y\dot{y} + 32\mu[\vec{r}\dot{\vec{r}}]}{r^{10}} \\
&- \frac{1890[\vec{r}\dot{\vec{r}}]^2 z\dot{z} + 945[\vec{r}\dot{\vec{r}}][\dot{\vec{r}}\dot{\vec{r}}]z^2 + 315[\vec{r}\dot{\vec{r}}]^3}{r^{11}} \\
&- \frac{240\mu[\vec{r}\vec{r}]z\dot{z} + 150\mu x\dot{x}z^2 + 200\alpha z\dot{z} + 15\alpha x\dot{x} + 180\mu[\vec{r}\dot{\vec{r}}][\vec{r}\vec{r}] + 760\mu[\vec{r}\dot{\vec{r}}]z^2 + 40\alpha[\vec{r}\dot{\vec{r}}]}{r^{12}} + \frac{3465[\vec{r}\dot{\vec{r}}]^3 z^2}{r^{13}} \\
&+ \frac{260\alpha[\vec{r}\vec{r}]z\dot{z} + 180\alpha x\dot{x}z^2 + 1200\alpha z^3\dot{z} + 1380\mu[\vec{r}\dot{\vec{r}}][\vec{r}\vec{r}]z^2 + 1520\alpha[\vec{r}\dot{\vec{r}}]z^2 + 130\alpha[\vec{r}\dot{\vec{r}}][\vec{r}\vec{r}]}{r^{14}} \\
&- \frac{700\alpha z^5\dot{z} + 1400\alpha[\vec{r}\vec{r}]z^3\dot{z} + 700\alpha y\dot{y}z^4 + 4340\alpha[\vec{r}\dot{\vec{r}}]z^4 + 1820\alpha[\vec{r}\vec{r}][\vec{r}\dot{\vec{r}}]z^2}{r^{16}} + \frac{5950\alpha[\vec{r}\dot{\vec{r}}][\vec{r}\vec{r}]z^4}{r^{18}}) \\
&\dot{x}(-\frac{15[\dot{\vec{r}}\dot{\vec{r}}] + 30\dot{z}^2}{r^7} - \frac{2\mu}{r^8} + \frac{105[\dot{\vec{r}}\dot{\vec{r}}]z^2 + 105[\vec{r}\dot{\vec{r}}]^2 + 420[\vec{r}\dot{\vec{r}}]z\dot{z}}{r^9} + \frac{24\mu[\vec{r}\vec{r}] + 58\mu z^2 + 32\mu x^2 + \alpha}{r^{10}} \\
&- \frac{945[\vec{r}\dot{\vec{r}}]^2 z^2}{r^{11}} - \frac{(15\alpha y^2 + 145\alpha z^2 + 150\mu z^4 + 150\mu y^2 z^2}{r^{12}} + \frac{565\alpha z^4 + 180\alpha y^2 z^2}{r^{14}} \\
&- \frac{525\alpha[\vec{r}\vec{r}]z^4 + 700\alpha x^2 z^4}{r^{16}})\}
\end{aligned}
$$

$$
\begin{aligned}
\frac{d^5y}{dt^5} &= y_{Kep,5} + \alpha\{ \\
&y(\frac{210[\vec{r}\dot{\vec{r}}]\dot{z}^2 + 210[\dot{\vec{r}}\dot{\vec{r}}]z\dot{z} + 105[\vec{r}\dot{\vec{r}}][\dot{\vec{r}}\dot{\vec{r}}]}{r^9} + \frac{136\mu z\dot{z} + 32\mu x\dot{x} + 32\mu[\vec{r}\dot{\vec{r}}]}{r^{10}} \\
&- \frac{1890[\vec{r}\dot{\vec{r}}]^2 z\dot{z} + 945[\vec{r}\dot{\vec{r}}][\dot{\vec{r}}\dot{\vec{r}}]z^2 + 315[\vec{r}\dot{\vec{r}}]^3}{r^{11}} \\
&- \frac{200\mu z^3\dot{z} + 200\mu x\dot{x}z^2 + 240\mu[\vec{r}\vec{r}]z\dot{z} + 180\mu[\vec{r}\vec{r}][\vec{r}\dot{\vec{r}}] + 560\mu[\vec{r}\dot{\vec{r}}]z^2 + 220\alpha z\dot{z} + 20\alpha x\dot{x} + 20\alpha[\vec{r}\dot{\vec{r}}]}{r^{12}} \\
&+ \frac{3465[\vec{r}\dot{\vec{r}}]^3 z^2}{r^{13}} \\
&+ \frac{260\alpha[\vec{r}\vec{r}]z\dot{z} + 180\alpha y\dot{y}z^2 + 1200\alpha z^3\dot{z} + 1380\mu[\vec{r}\dot{\vec{r}}][\vec{r}\vec{r}]z^2 + 1520\alpha[\vec{r}\dot{\vec{r}}]z^2 + 130\alpha[\vec{r}\dot{\vec{r}}][\vec{r}\vec{r}]}{r^{14}} \\
&- \frac{700\alpha z^5\dot{z} + 1400\alpha[\vec{r}\vec{r}]z^3\dot{z} + 700\alpha x\dot{x}z^4 + 4340\alpha[\vec{r}\dot{\vec{r}}]z^4 + 1820\alpha[\vec{r}\dot{\vec{r}}][\vec{r}\vec{r}]z^2}{r^{16}} + \frac{5950\alpha[\vec{r}\dot{\vec{r}}][\vec{r}\vec{r}]z^4}{r^{18}}) \\
&+\dot{y}(-\frac{15[\dot{\vec{r}}\dot{\vec{r}}] + 30z^2}{r^7} - \frac{2\mu}{r^8} + \frac{105[\dot{\vec{r}}\dot{\vec{r}}]z^2 + 105[\vec{r}\dot{\vec{r}}]^2 + 420[\vec{r}\dot{\vec{r}}]z\dot{z}}{r^9} + \frac{24\mu[\vec{r}\vec{r}] + 58\mu z^2 + 32\mu y^2 + \alpha}{r^{10}} \\
&- \frac{945[\vec{r}\dot{\vec{r}}]^2 z^2}{r^{11}} - \frac{15\alpha[\vec{r}\vec{r}] + 130\alpha z^2 + 20\alpha y^2 + 150\mu[\vec{r}\vec{r}]z^2 + 200\mu y^2 z^2}{r^{12}} + \frac{565\alpha z^4 + 180\alpha x^2 z^2}{r^{14}} \\
&- \frac{525\alpha[\vec{r}\vec{r}]z^4 + 700\alpha y^2 z^4}{r^{16}})\}
\end{aligned}
$$

$$
\begin{aligned}
\frac{d^5z}{dt^5} &= z_{Kep,5} + \alpha\{ \\
&z(\frac{315[\vec{r}\dot{\vec{r}}][\dot{\vec{r}}\dot{\vec{r}}] + 630[\vec{r}\dot{\vec{r}}]\dot{z}^2 + 315[\dot{\vec{r}}\dot{\vec{r}}]z\dot{z}}{r^9} + \frac{72\mu y\dot{y} + 72\mu x\dot{x} + 96\mu[\vec{r}\dot{\vec{r}}]}{r^{10}} \\
&- \frac{945[\vec{r}\dot{\vec{r}}]^3 + 2835[\vec{r}\dot{\vec{r}}]^2 z\dot{z} + 945[\vec{r}\dot{\vec{r}}][\dot{\vec{r}}\dot{\vec{r}}]z^2}{r^{11}} \\
&- \frac{420\mu[\vec{r}\dot{\vec{r}}][\vec{r}\vec{r}] + 560\mu[\vec{r}\dot{\vec{r}}]z^2 + 200\mu z^2(x\dot{x} + y\dot{y}) + 390\mu z\dot{z}(x^2+y^2) + 60\alpha(x\dot{x} + y\dot{y}) + 180\alpha[\vec{r}\dot{\vec{r}}]}{r^{12}} \\
&+ \frac{3465[\vec{r}\dot{\vec{r}}]^3 z^2}{r^{13}} \\
&+ \frac{1380\mu[\vec{r}\dot{\vec{r}}][\vec{r}\vec{r}]z^2 + 440\alpha y\dot{y}z^2 + 440\alpha x\dot{x}z^2 + 390\alpha[\vec{r}\dot{\vec{r}}][\vec{r}\vec{r}] + 2280\alpha[\vec{r}\dot{\vec{r}}]z^2}{r^{14}} \\
&- \frac{700\alpha x\dot{x}z^4 + 700\alpha y\dot{y}z^4 + 4340\alpha[\vec{r}\dot{\vec{r}}]z^4 + 3220\alpha[\vec{r}\dot{\vec{r}}][\vec{r}\vec{r}]z^2}{r^{16}} + \frac{5950\alpha[\vec{r}\dot{\vec{r}}][\vec{r}\vec{r}]z^4}{r^{18}}) \\
&+\dot{z}(-\frac{30\dot{z}^2 + 45[\dot{\vec{r}}\dot{\vec{r}}]}{r^7} - \frac{6\mu}{r^8} + \frac{315[\vec{r}\dot{\vec{r}}]^2}{r^9} + \frac{9\alpha + 234\mu z^2 + 54\mu[\vec{r}\vec{r}]}{r^{10}} - \frac{590\mu z^4 + 45\alpha[\vec{r}\vec{r}] + 630\alpha z^2}{r^{12}} \\
&+ \frac{2025\alpha z^4 + 690\alpha[\vec{r}\vec{r}]z^2}{r^{14}} - \frac{700\alpha z^6 + 1925\alpha[\vec{r}\vec{r}]z^4}{r^{16}})\}
\end{aligned}
$$

D.1 Richtigkeit der Ergebnisse

Im Folgenden werden die höheren Ableitungen auf ihre Genauigkeit überprüft. Dies soll einerseits die Richtigkeit der Ableitungen nachweisen und andererseits Anwendungsfälle demonstrieren.

D.1.1 Vergleich der Reihenkoeffizienten

In den folgenden Tabellen werden die Potenzreihenkoeffizienten ($a_0 ... a_n$) auf ihre Genauigkeit überprüft. Die erste Spalte enthält den Index der Koeffizienten. In der zweiten Spalte befindet sich der berechnete Koeffizientenwert, gebildet aus den höheren Ableitungen. Als Referenzlösung werden die Koeffizienten des Potenzreihenintegrators verwendet. Diese sind in der dritten Spalte eingetragen. Um die Qualität zu beurteilen, wurde der absolute Fehler (ε_{abs}) in die vierte Spalte und der relative Fehler (ε_{rel}) in die fünfte Spalte eingetragen.

i	a_i berechnet	a_i PowserInt	ε_{abs}	ε_{rel}
0	-5020.823835619	-5020.823835619	0	0
1	-5.150424898	-5.150424898	0	0
2	0.32128416175413326077e-2	0.32128416175413326077e-2	0.50e-50	0.17e-47
3	0.11054906183495459871e-5	0.11054906183495459871e-5	0.26e-53	0.24e-47
4	-0.33936694289394860835e-9	-0.33936694289394860835e-9	0.32e-57	0.94e-48
5	0.73999074230564116092e-13	-0.73999074230564116092e-13	0.93e-60	0.13e-46

Tabelle D.1: Vergleich der Reihenterme (X-Komponente)

i	a_i berechnet	a_i PowserInt	ε_{abs}	ε_{rel}
0	62.177071042	62.177071042	0	0
1	-0.25747324e-1	-0.25747324e-1	0	0
2	-0.39787311413592603205e-4	-0.39787311413592603205e-4	0.42e-52	0.11e-47
3	0.54064917061813322411e-8	0.54064917061813322411e-8	0.26e-55	0.47e-47
4	0.42641787555594467289e-11	0.42641787555594467289e-11	0.99e-59	0.23e-47
5	-0.30833344955564757635e-15	-0.30833344955564757635e-15	0.30e-62	0.99e-47

Tabelle D.2: Vergleich der Reihenterme (Y-Komponente)

i	a_i berechnet	a_i PowserInt	ε_{abs}	ε_{rel}
0	4547.497626756	4547.497626756	0	0
1	-5.680500488	-5.680500488	0	0
2	-0.29183516589613189055e-2	-0.29183516589613189055e-2	0.53e-50	0.18e-47
3	0.12089105654256080856e-5	0.12089105654256080856e-5	0.40e-53	0.32e-47
4	0.31626884457524858571e-9	0.31626884457524858571e-9	0.96e-57	0.30e-47
5	-0.74944174955735097825e-13	-0.74944174955735097825e-13	0.86e-60	0.11e-46

Tabelle D.3: Vergleich der Reihenterme (Z-Komponente)

Der Vergleich der Reihenterme zeigt, dass diese im Rahmen der hier verwendeten numerischen Genauigkeit übereinstimmen (siehe ε_{abs}). Die Zunahme des relativen Fehlers (ε_{rel}) in den höheren Reihentermen resultiert aus der erhöhten Anzahl an Rechenoperationen. Diese verursachen einen Zuwachs des Rundungsfehlers, welcher nicht vermieden werden kann.

D.1.2 Vergleich der Ortskoordinaten

Durch Einsetzen der höheren Ableitungen in eine Taylorreihe und deren Auswertung zu einem bestimmten Zeitpunkt erhält man eine weitere Möglichkeit, die Qualität der Ergebnisse zu beurteilen. Die berechneten Ortskoordinaten aus der Taylorreihe können anschließend mit einer Referenzlösung verglichen werden (ε_{abs}). In Abbildung D.1 wurde eine Bahn mit Hilfe der LiDIA-Bibliothek und dem Potenzreihenintegrator berechnet und mit der Taylorreihenlösung verglichen. Dabei zeigt sich, dass der absolute Fehler nach 60 Sekunden für die X und Z-Komponente im μm und für die Y-Komponente im 10 nm-Bereich liegt. Eine sukzessive Auswertung mit Hilfe der Taylorreihenlösung ist daher nur in sehr kurzen Schritten möglich. Da die Taylorreihe des Hauptproblems sehr langsam konvergiert, werden viele Reihenterme benötigt, um einen größeren Zeitraum in einem Schritt auszuwerten. Erschwerend kommt hinzu, dass die Kompliziertheit der analytischen Darstellung mit steigendem Grad der Ableitung zunimmt. Dies macht die Ausarbeitung weiterer Terme und deren anschließende Rücksubstitution zu einer nahezu unlösbaren Aufgabe, die lediglich durch Computer Algebra Systeme bewältigt werden kann. Vielversprechende Softwarepakete dazu sind

CAS Computer Algebra Systeme

MATHEMATICA[1], MAXIMA[2] und SymbolicC++[3].

Abbildung D.1: Absoluter Fehler in Meter [m]

D.1.3 Vergleich der Rechenzeit

Beim Rechenzeitvergleich (auf Testplattform 1, siehe Abschnitt A.1) zeigen sich die Stärken der analytischen Lösung gegenüber dem Potenzeihenintegrator. Bei der Berechnung einer Potenzreihe mit Hilfe des Potenzreihenintegrators fallen eine ganze Reihe von Funktionsaufrufen an, die Rechenzeit benötigen. Außerdem sind die Rechenoperationen zur Verknüpfung von Potenzreihen rekursiv definiert[4], was eine Parallelisierung durch den Compiler auf Assemblerebene nahezu unmöglich macht. Die Berechnung der analytisch erarbeiteten Lösung erfolgt deutlich schneller als die des Potenzreihenintegrators (siehe dazu D.2). Außerdem kann ein bestimmter Reihenterm ohne vorheriges Berechnen der vorangegangenen Ableitungen ermittelt werden. Dies ist der Rücksubstitution zu verdanken, da diese Lösung ausschließlich von Ort - und Geschwindigkeitsvektor abhängt.

[1] http://www.wolfram.com/
[2] http://maxima.sourceforge.net/
[3] http://issc.uj.ac.za/symbolic/symbolic.html
[4] siehe dazu auch [Wanner68] und [Schneider92]

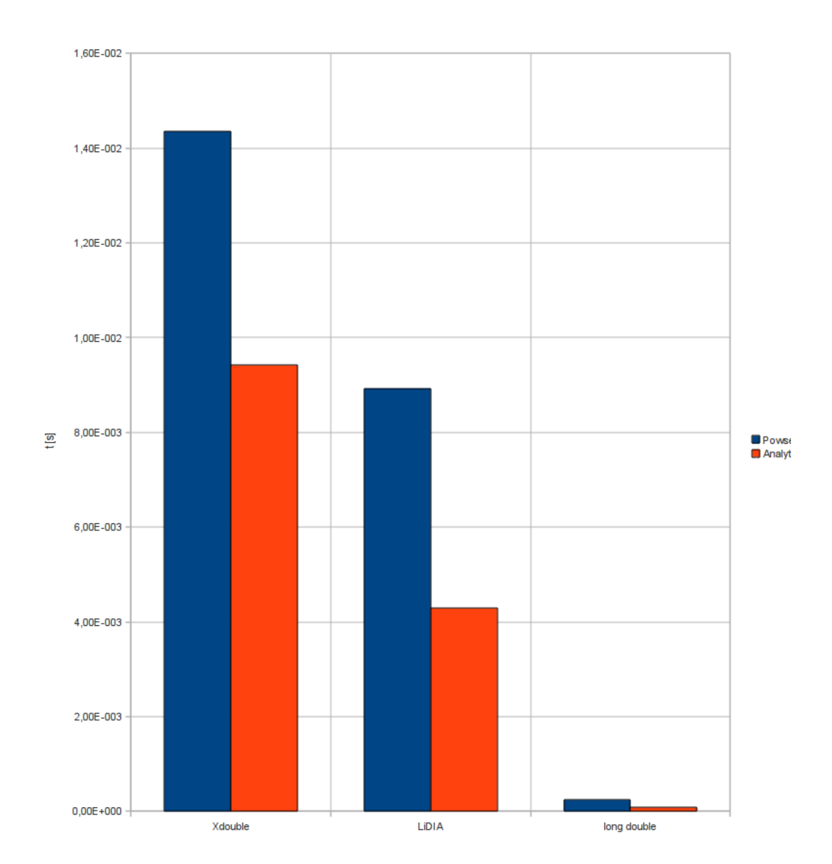


Abbildung D.2: Vergleich der Rechenzeit zur Lösung des Hauptproblems mit verschiedenen Datentypen. Der Xdouble-Datentyp berücksichtigt 64 signifikante Nachkommastellen; der LiDIA-Datentyp wurde auf 32 signifikante Nachkommastellen eingestellt und der long double - Datentyp verwendet definitionsgemäß 16 signifikante Nachkommastellen

Anhang E

Formeln zur numerischen Berechnung von höheren Ableitungen

Es wurden die folgenden Funktionswerte verwendet:

Zur Bestimmung der Lösungsformeln wurde das Computeralgebrasystem MATHEMATICA verwendet

$f(-9h+x_0), f(-8h+x_0), f(-7h+x_0), f(-6h+x_0)$
$f(-5h+x_0), f(-4h+x_0), f(-3h+x_0), f(-2h+x_0)$
$f(h+x_0), f(+x_0), f(h-x_0)$
$f(2h+x_0), f(3h+x_0), f(4h+x_0), f(5h+x_0)$
$f(6h+x_0), f(7h+x_0), f(8h+x_0), f(9h+x_0)$

Durch Auflösung des Gleichungssystems nach den jeweiligen Unbekannten $f^{'}$, ..., $f^{(18)}$ ergeben sich folgende Lösungsformeln:

$$\begin{aligned} f'(x_0) = {} & \frac{1}{12252240h}(-28f(-9h+x_0) + 567f(-8h+x_0) - 5508f(-7h+x_0) \\ & + 34272f(-6h+x_0) - 154224f(-5h+x_0) + 539784f(-4h+x_0) \\ & - 1559376f(-3h+x_0) + 4009824f(-2h+x_0) - 11027016f(-h+x_0) \\ & + 11027016f(h+x_0) - 4009824f(2h+x_0) + 1559376f(3h+x_0) \\ & - 539784f(4h+x_0) + 154224f(5h+x_0) - 34272f(6h+x_0) \\ & + 5508f(7h+x_0) - 567f(8h+x_0) + 28f(9h+x_0)) \\ & - \frac{h^{18}f^{(19)}(c)}{923780} \end{aligned}$$

$$\begin{aligned} f''(x_0) = {} & \frac{1}{15437822400h^2}(-47541321542f(x_0) + 7840f(-9h+x_0) \\ & - 178605f(-8h+x_0) + 1982880f(-7h+x_0) - 14394240f(-6h+x_0) \\ & + 77728896f(-5h+x_0) - 340063920f(-4h+x_0) + 1309875840f(-3h+x_0) \\ & - 5052378240f(-2h+x_0) + 27788080320f(-h+x_0) + 27788080320f(h+x_0) \\ & - 5052378240f(2h+x_0) + 1309875840f(3h+x_0) - 340063920f(4h+x_0) \\ & + 77728896f(5h+x_0) - 14394240f(6h+x_0) + 1982880f(7h+x_0) \\ & - 178605f(8h+x_0) + 7840f(9h+x_0)) \\ & - \frac{h^{18}f^{(20)}(c)}{9237800} \end{aligned}$$

$$
\begin{aligned}
f^{(3)}(x_0) = {} & \frac{1}{3027024000h^3}(63397f(-9h+x_0) - 1281033f(-8h+x_0) \\
& + 12405267f(-7h+x_0) - 76813928f(-6h+x_0) + 342868500f(-5h+x_0) \\
& - 1182036366f(-4h+x_0) + 3302404924f(-3h+x_0) - 7666346376f(-2h+x_0) \\
& + 8823005334f(-h+x_0) - 8823005334f(h+x_0) + 7666346376f(2h+x_0) \\
& - 3302404924f(3h+x_0) + 1182036366f(4h+x_0) - 342868500f(5h+x_0) \\
& + 76813928f(6h+x_0) - 12405267f(7h+x_0) + 1281033f(8h+x_0) \\
& - 63397f(9h+x_0)) + \frac{514639h^{16}f^{(19)}(c)}{51459408000}
\end{aligned}
$$

$$
\begin{aligned}
f^{(4)}(x_0) = {} & \frac{1}{54486432000h^4}(842762184550f(x_0) - 507176f(-9h+x_0) \\
& + 11529297f(-8h+x_0) - 127597032f(-7h+x_0) + 921767136f(-6h+x_0) \\
& - 4937306400f(-5h+x_0) + 21276654588f(-4h+x_0) - 79257718176f(-3h+x_0) \\
& + 275988469536f(-2h+x_0) - 635256384048f(-h+x_0) - 635256384048f(h+x_0) \\
& + 275988469536f(2h+x_0) - 79257718176f(3h+x_0) + 21276654588f(4h+x_0) \\
& - 4937306400f(5h+x_0) + 921767136f(6h+x_0) - 127597032f(7h+x_0) \\
& + 11529297f(8h+x_0) - 507176f(9h+x_0)) \\
& + \frac{514639h^{16}f^{(20)}(c)}{257297040000}
\end{aligned}
$$

$$
\begin{aligned}
f^{(5)}(x_0) = {} & \frac{1}{21794572800h^5}(-3739217f(-9h+x_0) + 75119112f(-8h+x_0) \\
& - 721271943f(-7h+x_0) + 4407497768f(-6h+x_0) - 19241468100f(-5h+x_0) \\
& + 63619040016f(-4h+x_0) - 161682804556f(-3h+x_0) + 275637687624f(-2h+x_0) \\
& - 246459164094f(-h+x_0) + 246459164094f(h+x_0) - 275637687624f(2h+x_0) \\
& + 161682804556f(3h+x_0) - 63619040016f(4h+x_0) + 19241468100f(5h+x_0) \\
& - 4407497768f(6h+x_0) + 721271943f(7h+x_0) - 75119112f(8h+x_0) \\
& + 3739217f(9h+x_0)) - \frac{364919h^{14}f^{(19)}(c)}{4358914560}
\end{aligned}
$$

$$
\begin{aligned}
f^{(6)}(x_0) = {} & \frac{1}{32691859200h^6}(-2697076863910f(x_0) + 3739217f(-9h+x_0) \\
& - 84509001f(-8h+x_0) + 927349641f(-7h+x_0) - 6611246652f(-6h+x_0) \\
& + 34634642580f(-5h+x_0) - 143142840036f(-4h+x_0) + 485048413668f(-3h+x_0) \\
& - 1240369594308f(-2h+x_0) + 2218132476846f(-h+x_0) + 2218132476846f(h+x_0) \\
& - 1240369594308f(2h+x_0) + 485048413668f(3h+x_0) - 143142840036f(4h+x_0) \\
& + 34634642580f(5h+x_0) - 6611246652f(6h+x_0) + 927349641f(7h+x_0) \\
& - 84509001f(8h+x_0) + 3739217f(9h+x_0)) - \frac{364919h^{14}f^{(20)}(c)}{14529715200}
\end{aligned}
$$

$$f^{(7)}(x_0) = \frac{1}{11404800h^7}(14037f(-9h+x_0) - 278998f(-8h+x_0)$$
$$+ 2637347f(-7h+x_0) - 15732348f(-6h+x_0) + 65988500f(-5h+x_0)$$
$$- 202775476f(-4h+x_0) + 443422524f(-3h+x_0) - 641013196f(-2h+x_0)$$
$$+ 510956534f(-h+x_0) - 510956534f(h+x_0) + 641013196f(2h+x_0)$$
$$- 443422524f(3h+x_0) + 202775476f(4h+x_0) - 65988500f(5h+x_0)$$
$$+ 15732348f(6h+x_0) - 2637347f(7h+x_0) + 278998f(8h+x_0)$$
$$- 14037f(9h+x_0)) + \frac{5839219h^{12}f^{(19)}(c)}{9340531200}$$

$$f^{(8)}(x_0) = \frac{1}{119750400h^8}(50145087850f(x_0) - 131012f(-9h+x_0)$$
$$+ 2929479f(-8h+x_0) - 31648164f(-7h+x_0) + 220252872f(-6h+x_0)$$
$$- 1108606800f(-5h+x_0) + 4258284996f(-4h+x_0) - 12415830672f(-3h+x_0)$$
$$+ 26922554232f(-2h+x_0) - 42920348856f(-h+x_0) - 42920348856f(h+x_0)$$
$$+ 26922554232f(2h+x_0) - 12415830672f(3h+x_0) + 4258284996f(4h+x_0)$$
$$- 1108606800f(5h+x_0) + 220252872f(6h+x_0) - 31648164f(7h+x_0)$$
$$+ 2929479f(8h+x_0) - 131012f(9h+x_0)) + \frac{5839219h^{12}f^{(20)}(c)}{23351328000}$$

$$f^{(9)}(x_0) = \frac{1}{806400h^9}(-6063f(-9h+x_0) + 118448f(-8h+x_0) - 1092217f(-7h+x_0)$$
$$+ 6276192f(-6h+x_0) - 24816700f(-5h+x_0) + 69370784f(-4h+x_0)$$
$$- 135969204f(-3h+x_0) + 178778336f(-2h+x_0) - 133953346f(-h+x_0)$$
$$+ 133953346f(h+x_0) - 178778336f(2h+x_0) + 135969204f(3h+x_0)$$
$$- 69370784f(4h+x_0) + 24816700f(5h+x_0) - 6276192f(6h+x_0)$$
$$+ 1092217f(7h+x_0) - 118448f(8h+x_0) + 6063f(9h+x_0))$$
$$- \frac{21713h^{10}f^{(19)}(c)}{5322240}$$

$$f^{(10)}(x_0) = \frac{1}{241920h^{10}}(-459622540f(x_0) + 2021f(-9h+x_0)$$
$$- 44418f(-8h+x_0) + 468093f(-7h+x_0) - 3138096f(-6h+x_0)$$
$$+ 14890020f(-5h+x_0) - 52028088f(-4h+x_0) + 135969204f(-3h+x_0)$$
$$- 268167504f(-2h+x_0) + 401860038f(-h+x_0) + 401860038f(h+x_0)$$
$$- 268167504f(2h+x_0) + 135969204f(3h+x_0) - 52028088f(4h+x_0)$$
$$+ 14890020f(5h+x_0) - 3138096f(6h+x_0) + 468093f(7h+x_0)$$
$$- 44418f(8h+x_0) + 2021f(9h+x_0))$$
$$- \frac{21713h^{10}f^{(20)}(c)}{10644480}$$

$$\begin{aligned}
f^{(11)}(x_0) = {} & \frac{1}{20160h^{11}}(757f(-9h+x_0) - 14408f(-8h+x_0) \\
& + 128107f(-7h+x_0) - 699088f(-6h+x_0) + 2573500f(-5h+x_0) \\
& - 6638576f(-4h+x_0) + 12076484f(-3h+x_0) - 14976656f(-2h+x_0) \\
& + 10816894f(-h+x_0) - 10816894f(h+x_0) + 14976656f(2h+x_0) \\
& - 12076484f(3h+x_0) + 6638576f(4h+x_0) - 2573500f(5h+x_0) \\
& + 699088f(6h+x_0) - 128107f(7h+x_0) + 14408f(8h+x_0) \\
& - 757f(9h+x_0)) + \frac{5473h^8 f^{(19)}(c)}{241920}
\end{aligned}$$

$$\begin{aligned}
f^{(12)}(x_0) = {} & \frac{1}{15120h^{12}}(109965350f(x_0) - 757f(-9h+x_0) \\
& + 16209f(-8h+x_0) - 164709f(-7h+x_0) + 1048632f(-6h+x_0) \\
& - 4632300f(-5h+x_0) + 14936796f(-4h+x_0) - 36229452f(-3h+x_0) \\
& + 67394952f(-2h+x_0) - 97352046f(-h+x_0) - 97352046f(h+x_0) \\
& + 67394952f(2h+x_0) - 36229452f(3h+x_0) + 14936796f(4h+x_0) \\
& - 4632300f(5h+x_0) + 1048632f(6h+x_0) - 164709f(7h+x_0) \\
& + 16209f(8h+x_0) - 757f(9h+x_0)) + \frac{5473h^8 f^{(20)}(c)}{403200}
\end{aligned}$$

$$\begin{aligned}
f^{(13)}(x_0) = {} & \frac{1}{480h^{13}}(-69f(-9h+x_0) + 1264f(-8h+x_0) \\
& - 10691f(-7h+x_0) + 54816f(-6h+x_0) - 188900f(-5h+x_0) \\
& + 458272f(-4h+x_0) - 792012f(-3h+x_0) + 945568f(-2h+x_0) \\
& - 667238f(-h+x_0) + 667238f(h+x_0) - 945568f(2h+x_0) \\
& + 792012f(3h+x_0) - 458272f(4h+x_0) + 188900f(5h+x_0) \\
& - 54816f(6h+x_0) + 10691f(7h+x_0) - 1264f(8h+x_0) \\
& + 69f(9h+x_0)) - \frac{619h^6 f^{(19)}(c)}{6048}
\end{aligned}$$

$$\begin{aligned}
f^{(14)}(x_0) = {} & \frac{1}{720h^{14}}(-15704260f(x_0) + 161f(-9h+x_0) \\
& - 3318f(-8h+x_0) + 32073f(-7h+x_0) - 191856f(-6h+x_0) \\
& + 793380f(-5h+x_0) - 2405928f(-4h+x_0) + 5544084f(-3h+x_0) \\
& - 9928464f(-2h+x_0) + 14011998f(-h+x_0) + 14011998f(h+x_0) \\
& - 9928464f(2h+x_0) + 5544084f(3h+x_0) - 2405928f(4h+x_0) \\
& + 793380f(5h+x_0) - 191856f(6h+x_0) + 32073f(7h+x_0) \\
& - 3318f(8h+x_0) + 161f(9h+x_0)) \\
& - \frac{619h^6 f^{(20)}(c)}{8640}
\end{aligned}$$

$$
\begin{aligned}
f^{(15)}(x_0) = \frac{1}{8h^{15}}(&3f(-9h + x_0) - 52f(-8h + x_0)\\
&+ 413f(-7h + x_0) - 1992f(-6h + x_0) + 6500f(-5h + x_0)\\
&- 15064f(-4h + x_0) + 25116f(-3h + x_0) - 29224f(-2h + x_0)\\
&+ 20306f(-h + x_0) - 20306f(h + x_0) + 29224f(2h + x_0)\\
&- 25116f(3h + x_0) + 15064f(4h + x_0) - 6500f(5h + x_0)\\
&+ 1992f(6h + x_0) - 413f(7h + x_0) + 52f(8h + x_0)\\
&- 3f(9h + x_0)) + \frac{17}{48}h^4 f^{(19)}(c)
\end{aligned}
$$

$$
\begin{aligned}
f^{(16)}(x_0) = \frac{1}{3h^{16}}(&135850f(x_0) - 2f(-9h + x_0) + 39f(-8h + x_0)\\
&- 354f(-7h + x_0) + 1992f(-6h + x_0) - 7800f(-5h + x_0)\\
&+ 22596f(-4h + x_0) - 50232f(-3h + x_0) + 87672f(-2h + x_0)\\
&- 121836f(-h + x_0) - 121836f(h + x_0) + 87672f(2h + x_0)\\
&- 50232f(3h + x_0) + 22596f(4h + x_0) - 7800f(5h + x_0)\\
&+ 1992f(6h + x_0) - 354f(7h + x_0) + 39f(8h + x_0)\\
&- 2f(9h + x_0)) + \frac{17}{60}h^4 f^{(20)}(c)
\end{aligned}
$$

$$
\begin{aligned}
f^{(17)}(x_0) = \frac{1}{2h^{17}}(&-f(-9h + x_0) + 16f(-8h + x_0)\\
&- 119f(-7h + x_0) + 544f(-6h + x_0) - 1700f(-5h + x_0)\\
&+ 3808f(-4h + x_0) - 6188f(-3h + x_0) + 7072f(-2h + x_0)\\
&- 4862f(-h + x_0) + 4862f(h + x_0) - 7072f(2h + x_0)\\
&+ 6188f(3h + x_0) - 3808f(4h + x_0) + 1700f(5h + x_0)\\
&- 544f(6h + x_0) + 119f(7h + x_0) - 16f(8h + x_0)\\
&+ f(9h + x_0)) - \frac{5}{6}h^2 f^{(19)}(c)
\end{aligned}
$$

$$
\begin{aligned}
f^{(18)}(x_0) = \frac{1}{h^{18}}(&-48620f(x_0) + f(-9h + x_0)\\
&- 18f(-8h + x_0) + 153f(-7h + x_0) - 816f(-6h + x_0)\\
&+ 3060f(-5h + x_0) - 8568f(-4h + x_0) + 18564f(-3h + x_0)\\
&- 31824f(-2h + x_0) + 43758f(-h + x_0) + 43758f(h + x_0)\\
&- 31824f(2h + x_0) + 18564f(3h + x_0) - 8568f(4h + x_0)\\
&+ 3060f(5h + x_0) - 816f(6h + x_0) + 153f(7h + x_0)\\
&- 18f(8h + x_0) + f(9h + x_0)) - \frac{3}{4}h^2 f^{(20)}(c)
\end{aligned}
$$

Anhang F

Auswertungsprotokolle, Statistiken und zusätzliche Ergebnisse

F.1 Auswertung des ACOVEA Testlaufs

Im Folgenden ist das ACOVEA Protokoll der Simulation Abschnitt 5.4.2 auf Seite 127 abgedruckt.

```
Acovea 5.1.1 (compiled Jul 24 2010 14:48:24)
Evolving Better Software

Invented by Scott Robert Ladd         (scott.ladd@coyotegulch.com)
            Coyote Gulch Productions  (http://www.coyotegulch.com)

   test application: TestDecimal.cpp
        test system: martin
 config description: g++  Pentium  (version 1.0.0)
 test configuration: ../config/g++4.3_pentium.acovea
     acovea version: 5.1.1
    evocosm version: 3.1.0
application version: g++-4.3 4.6.0

   # of populations: 4
    population size: 16
      survival rate: 10% (2)
     migration rate: 5% (1)
      mutation rate: 1%
     crossover rate: 100%
    fitness scaling: sigma
 generations to run: 2
 random number seed: 3384417609
       testing mode: speed

    test start time: 2011 Jun 23 13:07:00


------------------------------------------------------------
generation 1 begins

population  1: ................
population  2: ................
population  3: ................
population  4: ................

generation 1 complete, average fitness: 0.003646
------------------------------------------------------------
generation 2 begins

population  1: ................
population  2: ................
population  3: ................
population  4: ................

generation 2 complete, average fitness: 0.00226784

Acovea completed its analysis at 2011 Jun 23 13:12:48

Optimistic options:

                    -fno-cprop-registers  (1.522)
                           -fno-tree-dse  (1.522)
                           -fno-tree-ter  (1.522)
                       -fschedule-insns2  (2.158)
```

```
                         -falign-labels  (2.158)
                       -ftree-vectorize  (1.522)
                          -falign-loops  (1.522)
                          -ffloat-store  (2.793)
                           -fpeel-loops  (1.522)
                      -fno-function-cse  (1.522)
                     -ftree-loop-linear  (2.158)

Pessimistic options:

              -momit-leaf-frame-pointer  (-2.29)
          -fno-guess-branch-probability  (-1.655)
                     -fno-if-conversion  (-1.655)
                      -fstrict-aliasing  (-1.655)
                           -mfpmath=387  (-1.655)
                       -mfpmath=sse,387  (-2.29)
                               -msse4.1  (-2.29)
                               -msse4.2  (-2.29)
                    -D__NO_MATH_INLINES  (-1.655)

Acovea's Best-of-the-Best:
g++-4.3 -lrt -lm -O1 -march=nocona -fno-merge-constants -fno-thread-jumps -fno-delayed-branch -fno-tree-dse
-fno-tree-ter -fno-tree-lrs -foptimize-sibling-calls -frerun-loop-opt -fpeephole2 -fschedule-insns
-fschedule-insns2 -fsched-spec -fdelete-null-pointer-checks -freorder-blocks -funit-at-a-time -falign-jumps
-falign-loops -falign-labels -ftree-pre -fstrict-overflow -finline-functions -funswitch-loops -fgcse-after-reload
-fpredictive-commoning -falign-functions -falign-jumps -falign-labels -fprefetch-loop-arrays
-ftree-vect-loop-version -ffloat-store -fprefetch-loop-arrays -funswitch-loops -funroll-loops
-fbranch-target-load-optimize -fno-function-cse -fgcse-sm -freschedule-modulo-scheduled-loops -ftree-loop-linear
-ftree-loop-im -ftree-loop-ivcanon -ftree-vectorize -mieee-fp -mno-push-args -maccumulate-outgoing-args
-mno-align-stringops -finline-limit=800 -o /tmp/ACOVEA7D244FF9 TestDecimal.cpp

Acovea's Common Options:
g++-4.3 -lrt -lm -O1 -march=nocona -foptimize-sibling-calls -fschedule-insns2 -falign-labels -fstrict-overflow
-funswitch-loops -ffloat-store -fprefetch-loop-arrays -fno-function-cse -fgcse-sm -ftree-loop-linear -mieee-fp
-maccumulate-outgoing-args -o /tmp/ACOVEA25E3B7F3 TestDecimal.cpp

-O1:
g++-4.3 -lrt -lm -O1 -march=nocona -o /tmp/ACOVEA4C799B19 TestDecimal.cpp

-O2:
g++-4.3 -lrt -lm -O2 -march=nocona -o /tmp/ACOVEAB02E1C7D TestDecimal.cpp

-O3:
g++-4.3 -lrt -lm -O3 -march=nocona -o /tmp/ACOVEA3A091C62 TestDecimal.cpp

-Os:
g++-4.3 -lrt -lm -Os -march=nocona -o /tmp/ACOVEACFD776F7 TestDecimal.cpp


A relative graph of fitnesses:

     Acovea's Best-of-the-Best: *************************                              (0.001716)
       Acovea's Common Options: *****************************                          (0.001955)
                           -O1: **************************************************     (0.003309)
                           -O2: **************************************                 (0.002548)
                           -O3: ************************************************       (0.003172)
                           -Os: *****************************************              (0.002714)

Acovea is done.
```

F.2 Anzahl der aktiven GCC Optimierungsoptionen bei bestimmten Optimierungsstufen

Version	*O0*	*O1*	*O2*	*O3*	*Os*	Gesamt
gcc-3.3	12	11	37	39	43	74
gcc-3.4	10	9	39	43	45	87
gcc-4.0	17	18	46	49	53	116
gcc-4.1	19	19	46	49	54	126
gcc-4.2	19	19	46	49	54	126
gcc-4.3	56	56	77	82	74	136
gcc-4.4	55	55	80	87	80	155
gcc-4.5	68	68	92	98	93	166
gcc-4.6	30	30	62	68	70	192
gcc-4.7	30	30	63	69	71	193

Tabelle F.1: Übersicht zur Anzahl der aktivierten GCC Optimierungsoptionen

F.3 Konfigurationsdateien

F.3.1 Lorenz.conf

```
1  <ODE>
2      <InitialValues>
3          Name = Experiment1
4          x    =  10
5          y    =  20
6          z    =  30
7      </InitialValues>
8      <Parameter>
9          a  =  10
10         b  =  28
11         c  =  2.6666666666666666666666
12     </Parameter>
13 </ODE>
```

Listing F.1: Die Konfigurationsdatei: *Lorenz.conf*

F.4 Ergebnisse der Keplerbahnsimulation

Hier werden noch weitere Ergebnisse zu den Experimenten auf Seite 54 angegeben. In den Abbildungen F.1, F.2 und F.3 ist der absolute Fehler für die jeweiligen Simulationszeiträume aufgetragen.

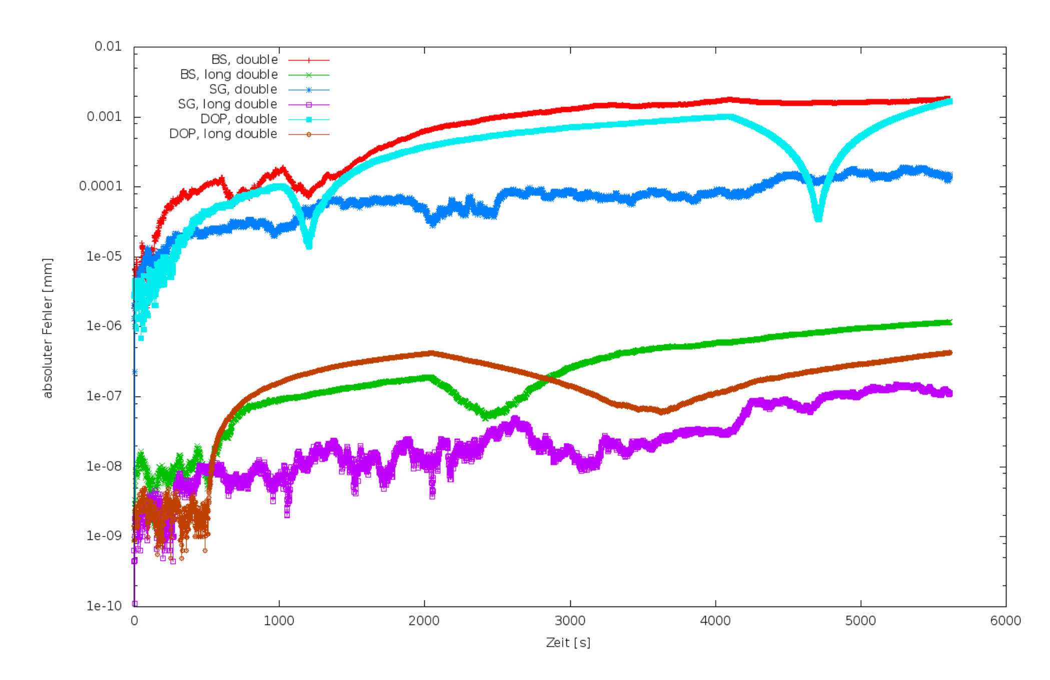


Abbildung F.1: Absoluter Fehler über den Simulationszeitraum eines Umlaufs für den double und long double Datentyp

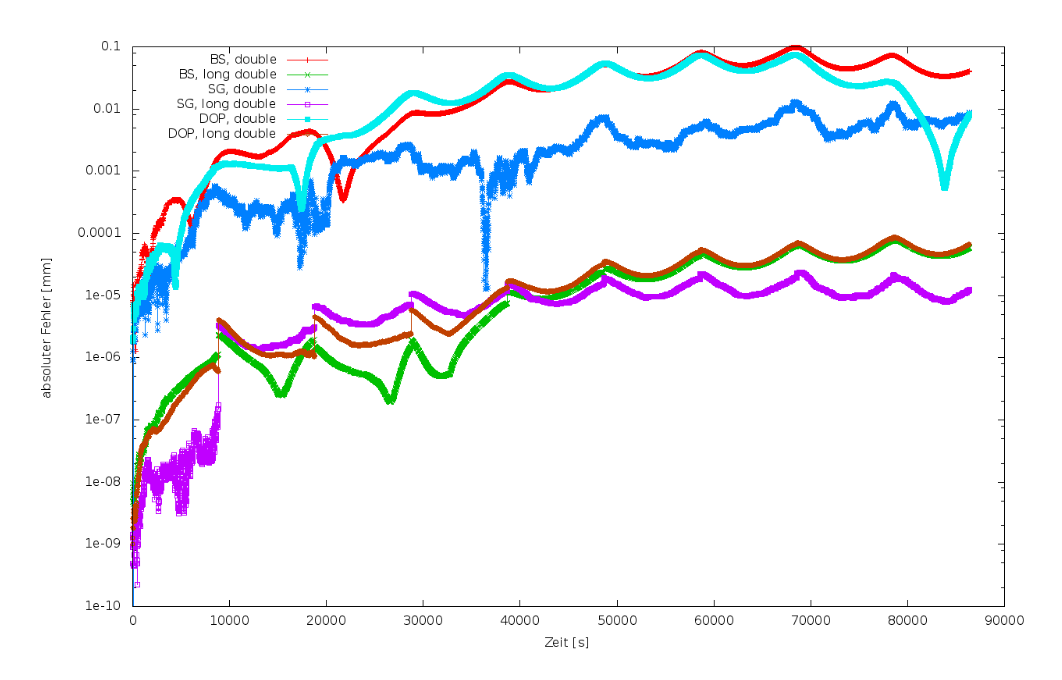


Abbildung F.2: Absoluter Fehler über den Simulationszeitraum eines Tages für den double und long double Datentyp

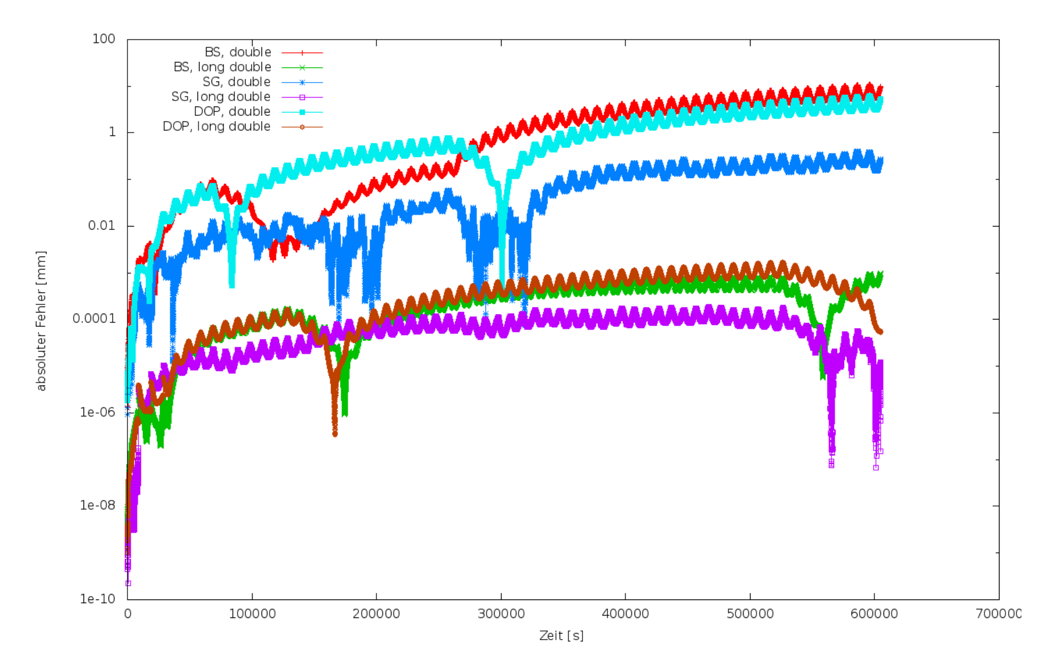


Abbildung F.3: Absoluter Fehler über den Simulationszeitraum von sieben Tagen für den double und long double Datentyp

F.5 Ergebnisse der Leistungsmessungen

F.5.1 Basisfunktionen

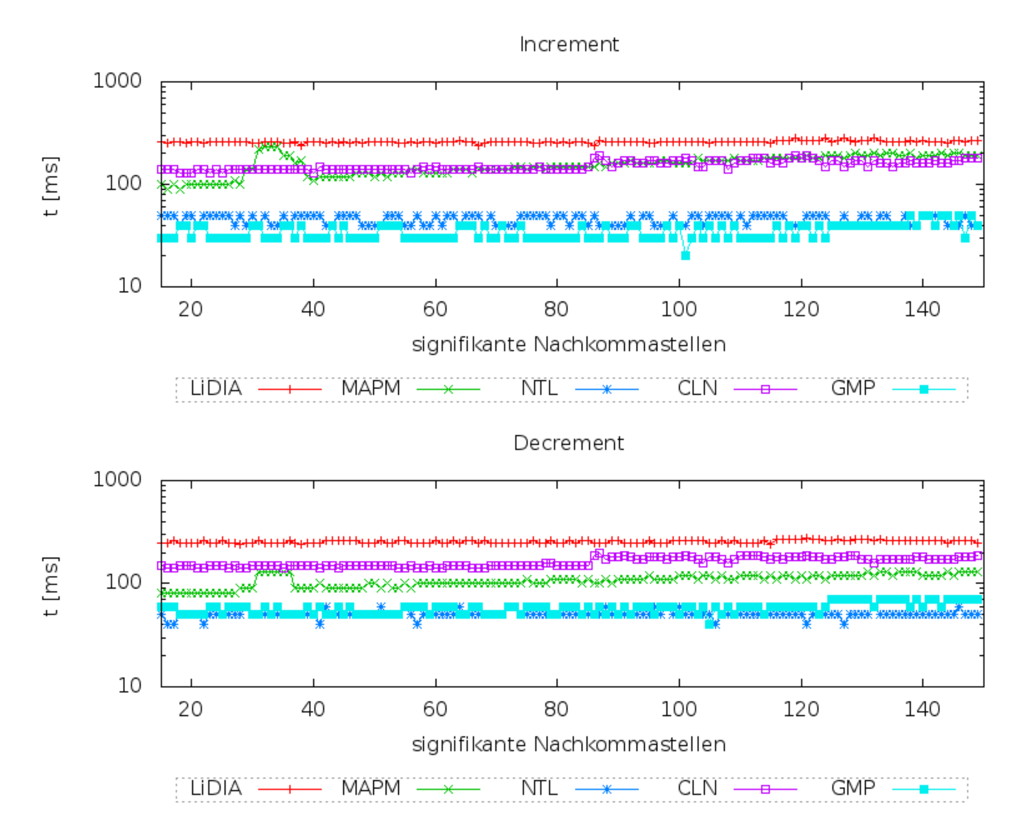


Abbildung F.4: Leistungsmessung zu Inkrement und Dekrement Operationen auf Testplattform 1

F.5.2 Exponentialfunktion

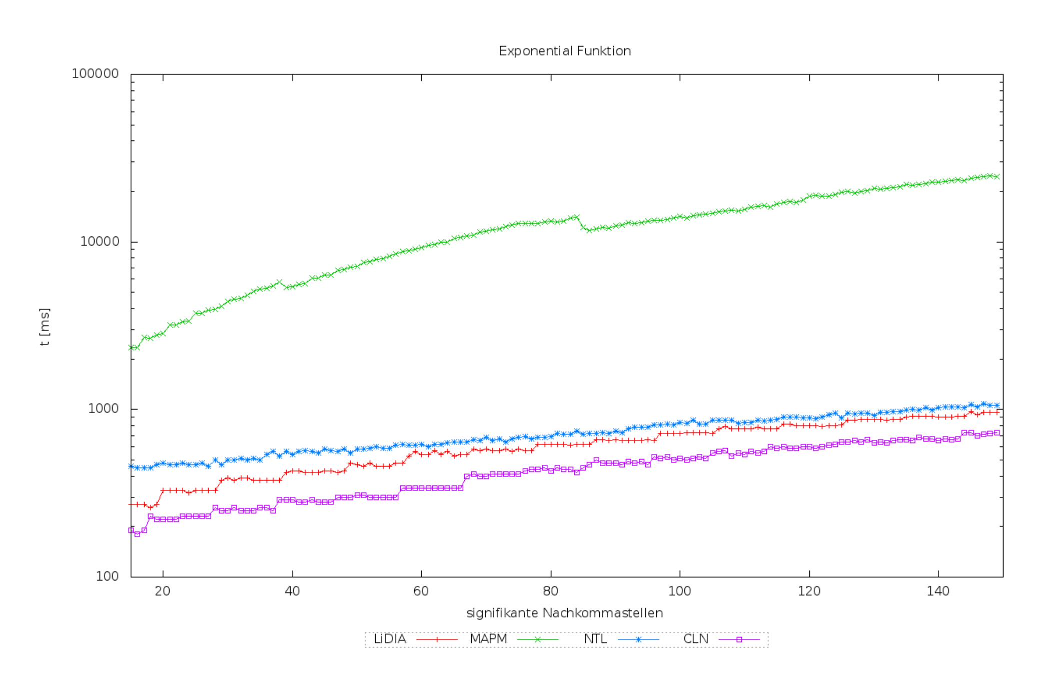


Abbildung F.5: Messergebnisse zur Exponentialfunktion (Testplattform 1)

F.5.3 Quadratwurzel

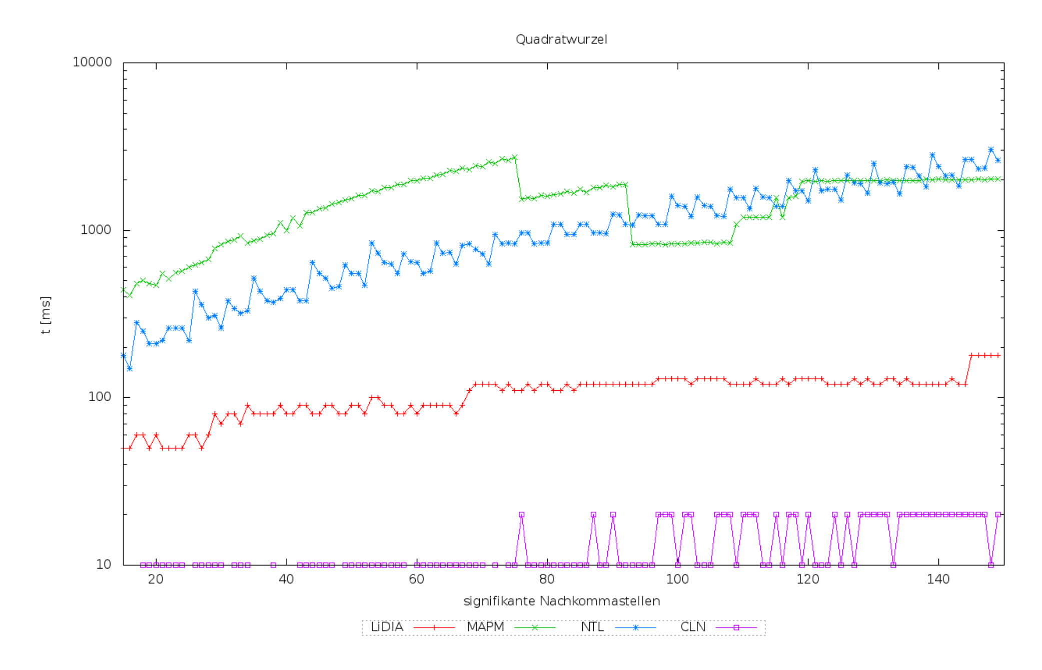


Abbildung F.6: Messergebnisse zur Quadratwurzel (Testplattform 1)

F.5.4 Sinus Hyperbolicus

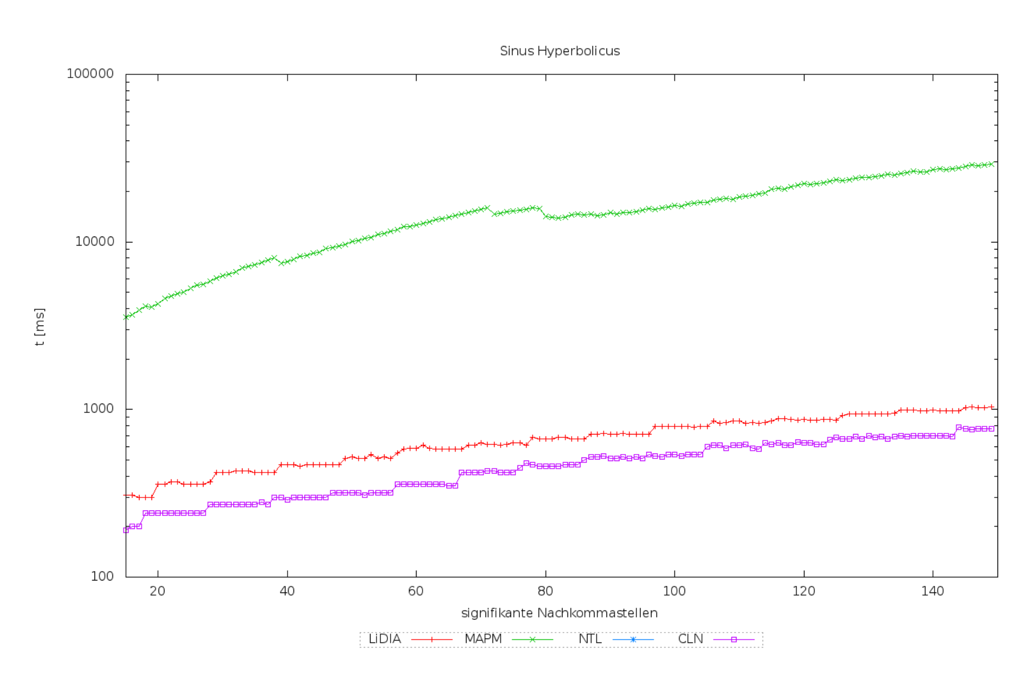


Abbildung F.7: Messergebnisse der *sinh*-Funktion (Testplattform 1)

F.5.5 Kosinus Hyperbolicus

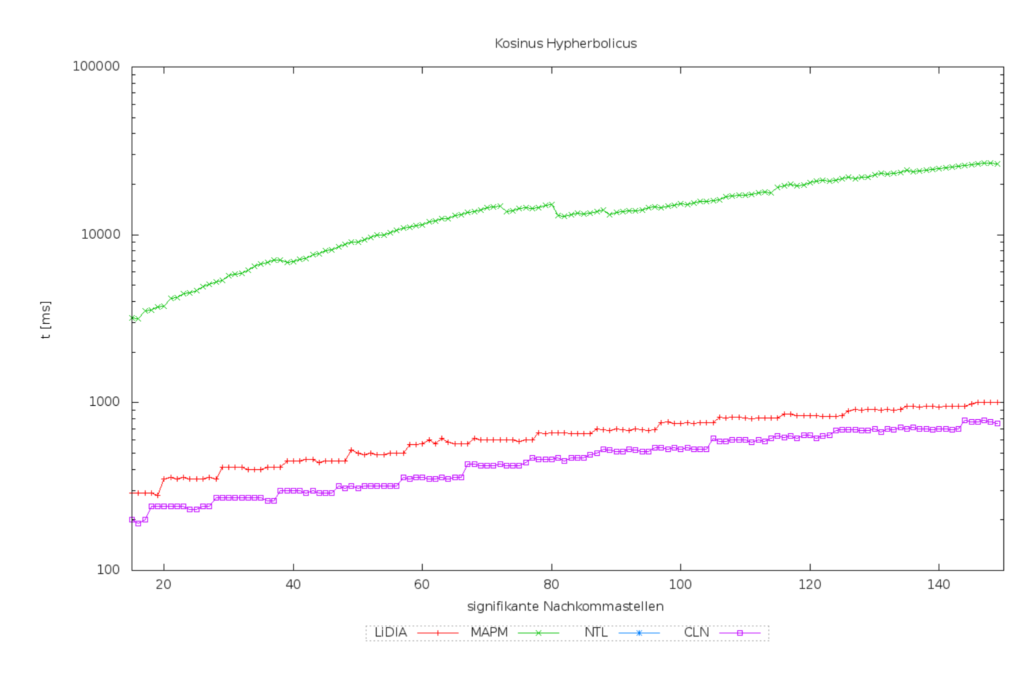


Abbildung F.8: Messergebnisse der *cosh*-Funktion (Testplattform 1)

F.5.6 Tangens Hyperbolicus

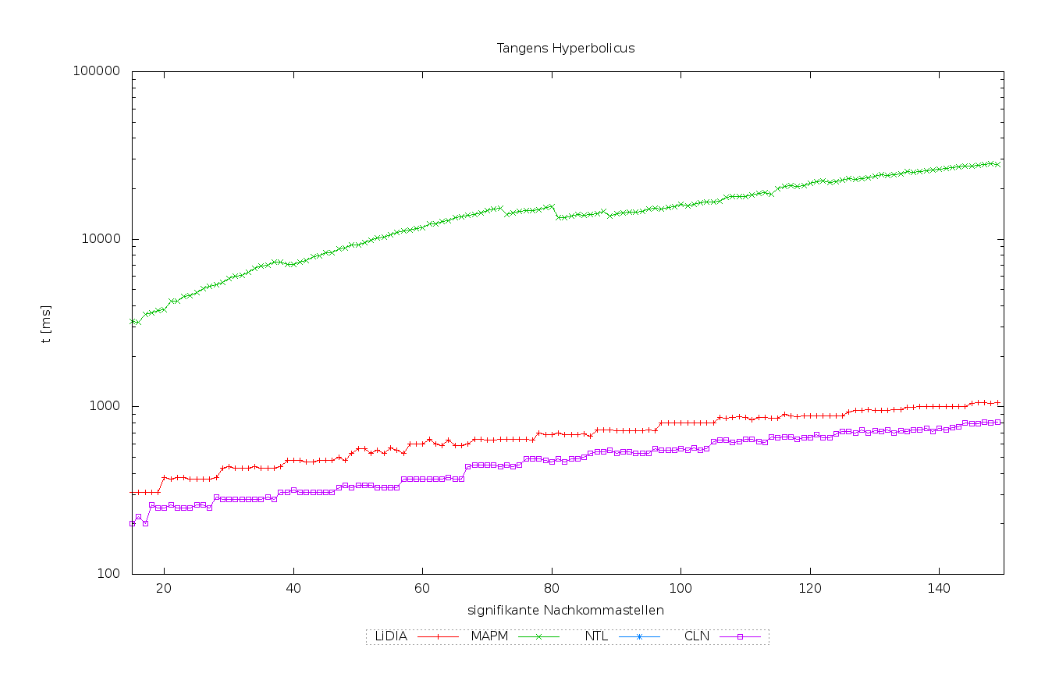


Abbildung F.9: Messergebnisse der *tan*-Funktion (Testplattform 1)

F.5.7 Arcus Sinus

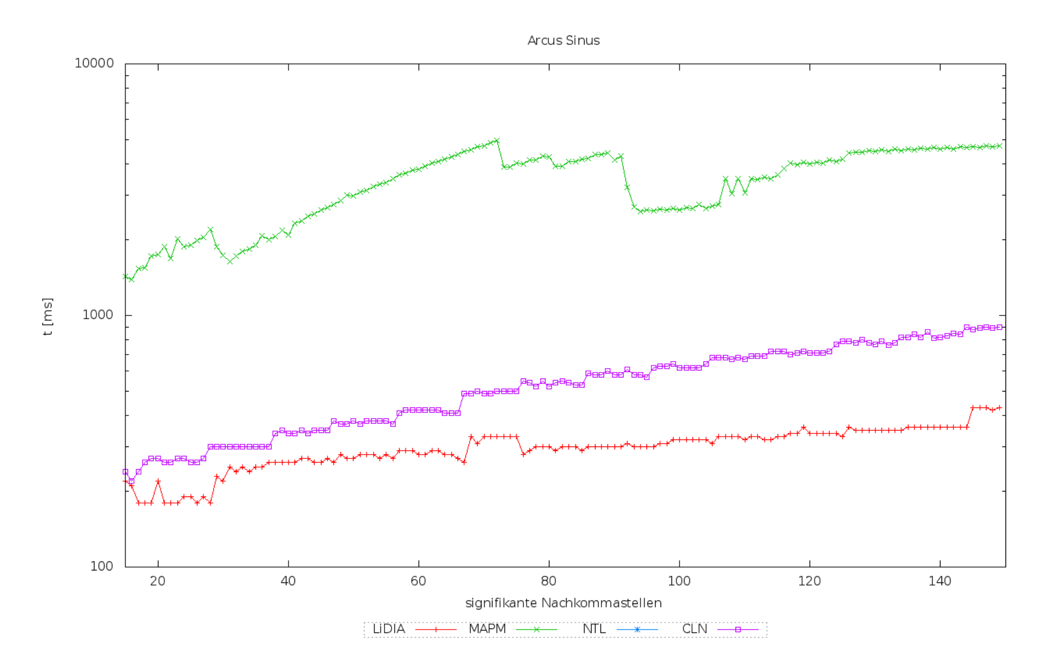


Abbildung F.10: Messergebnisse der *asin*-Funktion (Testplattform 1)

F.5.8 Arcus Kosinus

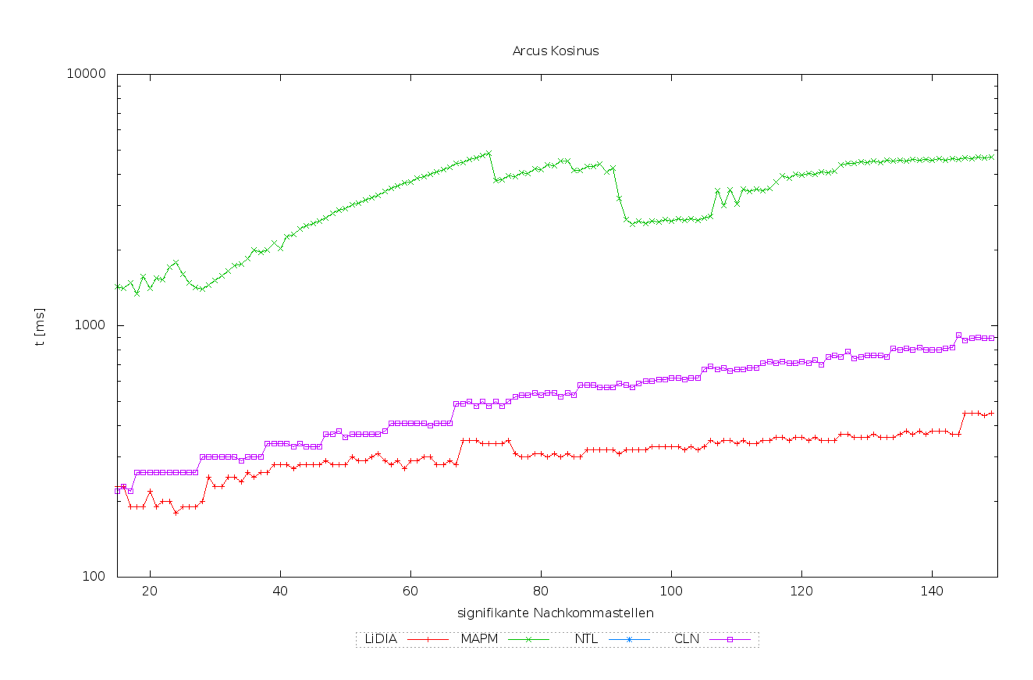


Abbildung F.11: Messergebnisse der *acos*-Funktion (Testplattform 1)

F.5.9 Arcus Tangens

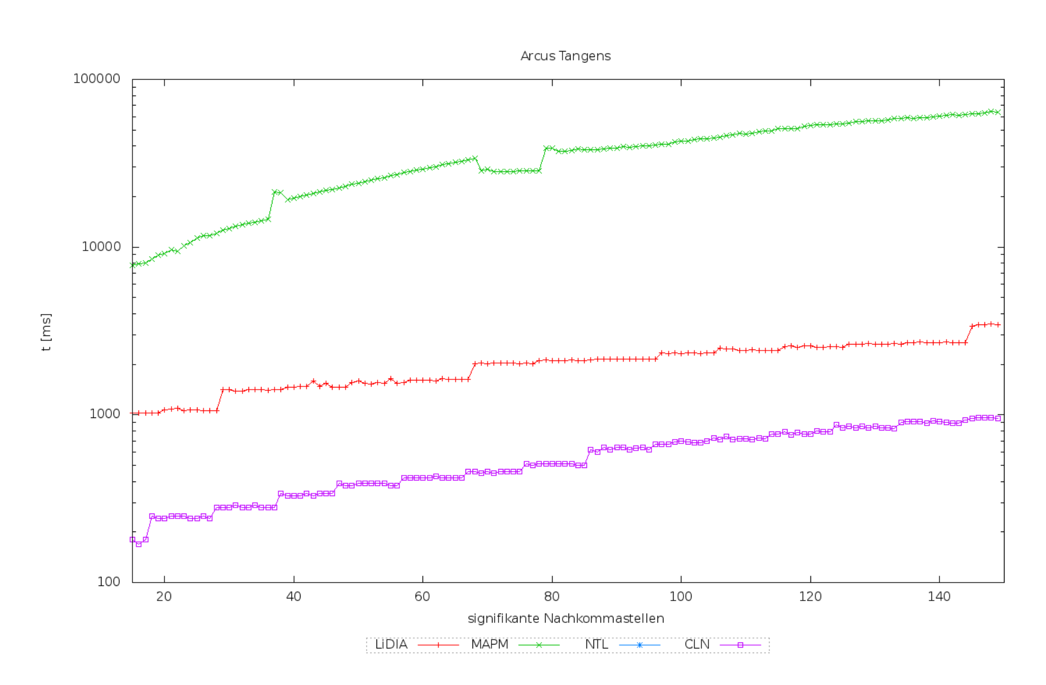


Abbildung F.12: Messergebnisse der *atan*-Funktion (Testplattform 1)

F.6 Vergleich zweier Lösungen des Erde-Mond Abstands

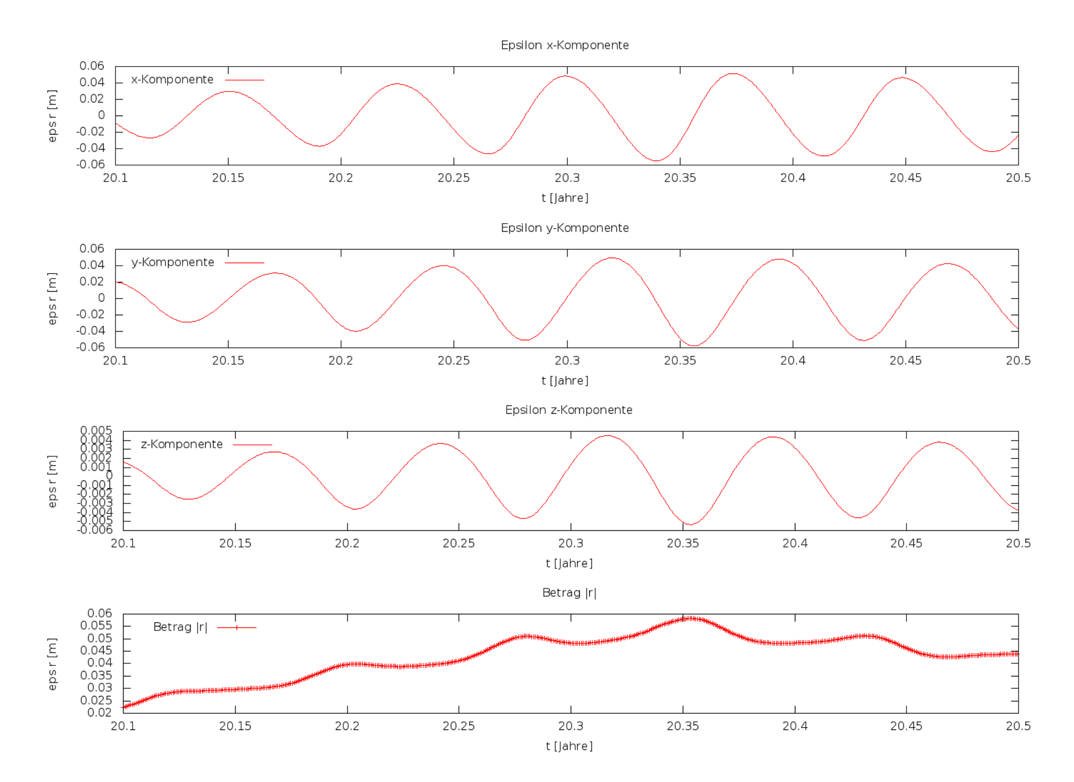


Abbildung F.13: Vergleich zweier Lösungen des Erde-Mond Abstands (Teilausschnitt)

Anhang G

Programme

G.1 TMP Vorlagen

G.1.1 Eine einfache TMP-*for*-Schleife als Klassentemplate

```
1       template< int iIndex > class For
2       {
3           public:
4               static inline void Loop()
5               {
6                   For< iIndex-1 >::Loop();
7               }
8       };
9       template <> class For< 0 >
10      {
11          public:
12              static inline void Loop()
13              {}
14      };
```

Listing G.1: TMP-*for*-Schleife

G.1.2 Eine zweifach geschachtelte TMP-*for*-Schleife als Klassentemplate

```
1 template< int i > class InnerFor
2 {
3     public:
4         static inline void Loop()
5         {
6             InnerFor< i-1 >::Loop();
7         }
8 };
9 template <> class InnerFor< 0 >
10 {
11     public:
12         static inline void Loop()
13         {}
14 };
15
```

```
16 template< int i > class For
17 {
18     public:
19         static inline void Loop()
20         {
21             For< i-1 >::Loop();
22             InnerFor < i >::Loop();
23         }
24 };
25 template <> class For< 0 >
26 {
27     public:
28         static inline void Loop()
29         {}
30 };
```

Listing G.2: TMP-*for*-Schleife

G.1.3 Verschiedene TMP-*while*-Schleifen

In diesem Abschnitt werden zwei verschiedene Methoden zur Umsetzung einer TMP-*while*-Schleife angegeben, wobei die Methode in Listing G.3 als Klassentemplate und die Methode in Listing G.4 als Funktionstemplate realisiert sind.

```
 1 template<int I> class While
 2 {
 3     public:
 4         static inline void Run()
 5         {
 6             While< ( (I-1) != 0 ) ? (I-1) : 0>::Run();
 7         }
 8 };
 9 template <> class While<0>
10 {
11     public:
12         static inline void Run(){}
13 };
```

Listing G.3: TMP-*while*-Schleife realisiert als Klassentemplate

```
1 template <int I> inline void While()
2 {
3    While< ( (I-1) != 0 ) ? (I-1) : 0 >();
4 }
5 template <> inline void While<0>(){}
```

Listing G.4: TMP-*while*-Schleife realisiert als Funktionstemplate

G.1.4 TMP-*if-else*-Struktur

```
1 template< bool Condition > class If
2 {
3     public:
4         static inline void Run()
```

```
5           {
6             // true
7           }
8 };
9 template <> class If< false >
10 {
11      public:
12           static inline void Run()
13           {
14             // false
15           }
16 };
```

Listing G.5: TMP-*if-else*-Struktur realisiert als Klassentemplate

```
1 template <bool Condidition> inline void If()
2 {
3      // true
4 }
5 template <> void If<false>()
6 {
7      // false
8 }
```

Listing G.6: TMP-*if-else*-Struktur realisiert als Funktionstemplate

G.1.5 TMP-*switch-case* Kontrollstruktur

```
1 enum CaseType
2 {
3      CASE1, CASE2
4 };
5
6 template< CaseType > class Switch
7 {
8      public:
9      static inline void Eval()
10     {
11          // default
12     }
13 };
14 template <> class Switch< CASE1 >
15 {
16     public:
17     static inline void Eval()
18     {
19          // Fall 1
20     }
21 };
22 template <> class Switch< CASE2 >
23 {
24     public:
25     static inline void Eval()
26     {
27          // Fall 2
```

```
28      }
29 };
```

Listing G.7: TMP-*switch-case*-Struktur realisiert als Klassentemplate

```
1  enum CaseType
2  {
3      CASE1,CASE2
4  };
5
6  template <CaseType> inline void Switch()
7  {
8      // default
9  }
10 template <> inline void Switch<CASE1>()
11 {
12     // Fall 1
13 }
14 template <> inline void Switch<CASE2>()
15 {
16     // Fall 2
17 }
```

Listing G.8: TMP-*switch-case*-Struktur realisiert als Funktionstemplate

G.1.6 TMP-Funktion zur Berechnung der Fakultät

Im Folgenden ist ein Programm zur Berechnung der Fakultät angegeben. Hierbei wird die Berechnung der Fakultät vollständig während der Übersetzungsphase durchgeführt.

```
1 template<unsigned long N, typename T> inline T Fact(void)
2 {
3   return (N==0) ? 1 : N*Fact<(N > 0 ? N-1 : 0),T>();
4 }
```

Listing G.9: C++-TMP-Funktion zur Berechnung der Fakultät

G.1.7 TMP-Funktion zur Potenzberechnung

Im Folgenden ist das Programm zu Potenzberechnung dargestellt. Hierfür wird eine weitere Hilfsfunktion Sqr benötigt. Diese berechnet das Quadrat aus einer gegebenen Gleitkommazahl.

```
1
2  template<typename T> static inline T Sqr(const T &x)
3  {
4    return x*x;
5  }
6
7  template <int P, typename T> inline T Pow(const T &x)
8  {
9    return (P == 0) ? 1
10          : ((P<0) ? 1/Pow<(P<0)?-P:0,T>(x)
11          : ((P%2) ? x*Sqr(Pow<(P%2)
```

```
12                          ? (P-1)/2 : 0,T>(x))
13                           : Sqr(Pow<(P%2) ? 0 : P/2,T>(x)))
14                  );
15 }
```

Listing G.10: C++-TMP-Funktion zur Potenzberechnung

G.2 Automatisches Differenzieren

G.3 Resultat aus Berechnung des Rückwärtsmodus

```
1 void dKepler(double *x, double *xb, double *y, double *yb, double *z,
      double
2         *zb, double *vx, double *vxb, double *vy, double *vyb, double
              *vz,
3         double *vzb) {
4     double r, _x, _y, _z;
5     double rb, _xb, _yb, _zb;
6     double tempb2;
7     double tempb1;
8     double tempb0;
9     double tempb;
10    _x = *x;
11    _y = *y;
12    _z = *z;
13    r = sqrt(_x*_x + _y*_y + _z*_z);
14    pushreal8(r);
15    r = r*(r*r);
16    tempb2 = -(0.3986004415000000E+6*(*vxb)/r);
17    tempb1 = -(0.3986004415000000E+6*(*vyb)/r);
18    tempb = -(0.3986004415000000E+6*(*vzb)/r);
19    rb = -(_y*tempb1/r) - _x*tempb2/r - _z*tempb/r;
20    *vzb = *zb;
21    *vyb = *yb;
22    *vxb = *xb;
23    popreal8(&r);
24    rb = 3*(r*r)*rb;
25    if (_x*_x + _y*_y + _z*_z == 0.0)
26        tempb0 = 0.0;
27    else
28        tempb0 = rb/(2.0*sqrt(_x*_x+_y*_y+_z*_z));
29    _zb = 2*_z*tempb0 + tempb;
30    _yb = 2*_y*tempb0 + tempb1;
31    _xb = 2*_x*tempb0 + tempb2;
32    *zb = _zb;
33    *yb = _yb;
34    *xb = _xb;
35 }
```

Listing G.11: Resultat aus Berechnung des Rückwärtsmodus

Abbildungsverzeichnis

Abkürzungsverzeichnis

AB Anfangsbedingungen

ABM Adams Bashforth Moulton

AD Automatisches Differenzieren

ACOVEA Analysis of Compiler Options via Evolutionary Algorithm

ALU Arithmetic Logical Unit

API Application Programming Interface

ASCII American Standard Code for Information Interchange

AW Anfangswerte

AWP Anfangswert Problem

BCH Baker Campbell Hausdorff

BKG Bundesamt für Kartographie und Geodäsie

BS Burlisch Stoer

DGL Differential Gleichung

CLI Command Line Interface

CLN Class Library for Numbers

CPU Central Processing Unit

EGM Earth Gravity Modell

EGM96 Earth Gravity Modell aus dem Jahr 1996

EGM08 Earth Gravity Modell aus dem Jahr 2008

EOP Earth Orientation Parameter

FMA fused multiplication and addition

FMS fused multiplication and subtraction

FPGA field-programmable gate array

FSW Fundamental Station Wettzell

GA Genetischer Algorithmus

GCC GNU Compiler Collection

GNU GNU is Not Unix

GMP GNU Multiprecision Library

GPS Global Positioning System

GPU Graphics Processing Unit

GRACE Gravity Recovery And Climate Experiment

GUI Graphical User Interface

IDL Interface Description Language

LAGEOS LAser GEOdynamics Satellite

LLR Lunar Laser Ranging

MAPM My Arbitrary Precision Math

MIT Massachusetts Institute of Technology

MMID Modified Midpoint Method

MPI Message Passing Interface

NTL Number Theorie Library

ODE Ordinary Differential Equation

OpenACC Open Accelerator

OpenMPI Open Message Passing Interface

OpenMP Open Multi-Processing

OOP Objektorientierte Programmierung

PCC Portable C Compiler

QUEST Quantum Engineering and Space Time Research

RAD Rapid Application Development

RAM Random Access Memory

RWP Randwert Problem

SG Shampine Gordon

SI Symplektischer Integrator

SLR Sattellite Laser Ranging

SPMD Single Programm Multiple Data

STL Standard Template Library

SWS Schrittweitensteuerung

TCC Tiny C Compiler

TMP Templatemetaprogrammierung

TUM Technische Universität München

UML Unified Modelling Language

XML Extensible Markup Language

Literaturverzeichnis

[Agner] Agner Fog : Homepage, Copenhagen University College of Engineering, http://www.agner.org/, Download am 01.05.2007

[Agner11] Agner Fog : **Optimizing software in C++: An optimization guide for Windows, Linux and Mac platforms**, http://www.agner.org/optimize, Download 01.05.2011

[Aho95-1] Aho Alfred V., Sethi Ravi, Ullmann Jeffrey D. : **Compilerbau, Teil 1**, Addison Wesley, 1995, 4. unveränderter Nachdruck, ISBN 3-89319-150-X

[Aho95-2] Aho Alfred V., Sethi Ravi, Ullmann Jeffrey D. : **Compilerbau, Teil 2**, Addison Wesley, 1995, 3. unveränderter Nachdruck, ISBN 3-89319-151-8

[Ajay08] Ajay D. Kshemkalyani: **Distributed Computing Principles, Algorithms, and Systems**, Cambridge University Press , 2008, ISBN-13 978-0-511-39341-9

[Alexandrescu01] Alexandrescu, Andrei:**Modern C++ Design: Generic Programming and Design Patterns Applied**, Addison Wesley, 2001, ISBN : 0-201-70431-5

[Auperle02] Aupperle Martin : **Die Kunst der Programmierung mit C++ - Exakte Grundlagen für die professionelle Softwareentwicklung**, 2. Auflage, 2002, Vieweg Verlag, ISBN 3-528-15481-0

[Axiom] Axiom Webseite: http://www.axiom-developer.org/, Download am 01.03.2011

[Backer03] Backer Reiner : **Assembler: Maschinennahes Programmieren von Anfang an. Mit Windows-Programmierung**, rororo Verlag, 3. Auflage, 2003, ISBN-10: 3499612240, ISBN-13: 978-3499612244

[Bansal06] Bansal Sorav, Aiken Alex : **Automatic Generation of Peephole Superoptimizers**, Computer Systems Lab, Stanford University, 2006, http://theory.stanford.edu/ sbansal/pubs/asplos06.pdf, Download am 07.07.2011

[Bauer89] Bauer, R.: **Bestimmung von Parametern des Erde-Mond-Systems - Ein Beitrag zur Modellerweiterung und Bewertung, Ergebnisse -**, Deutsche Geodätische Kommission, Reihe C, Nr. 353, München 1989

[Berry04] Berry Matthew M., Healy Liam M.: **Implementation of Gauss-Jackson Integration for Orbit Propagation**, in: The Journal of the Astronautical Sciences, Vol. 52, No. 3, July–September 2004, pp. 331–357

[blitz++] Webseite des *blitz++*-Projekts : http://www.oonumerics.org/blitz/?guid=ON, Download am 06.07.2011

[Brain06] Brain Martin, Crick Tom, De Vos Marina, Fitch John: **TOAST: Applying Answer Set Programming to Superoptimisation**, Lecture Notes in Computer Science, 2006, Volume 4079-2006, 270-284

[Brennecke00] Brennecke, Andreas: **Physikalische Analogien und Ziffernrechenmaschinen – Von mechanischen Rechengeräten zu Integrieranlagen und programmgesteuerten Maschinen**, Proceedings des 1. Symposiums zur **Entwicklung der Rechentechnik** vom 15. bis 17. September 2000 am Institut für Mathematik und Informatik der Ernst-Moritz-Arndt-Universität Greifswald, S. 89–111

[Bronstein08] Bronstein, I.N., Semendjajew, K.A.: **Taschenbuch der Mathematik**, Fünfte Auflage, Springer Verlag, ISBN 978-3-8171-2007-9

[Bücker06] Bücker Martin, Corliss George, Hovland Paul, Naumann Uwe, Norris Boyana: **Automatic Differentiation: Applications, Theory, and Implementations**, Springer-Verlag, 2006, ISBN: 3540284036

[Bulka99] Bulka Dov, Mayhew David : **Efficient C++: Performance Programming Techniques**, Addison-Wesley Professional, 1 Edition, 13. November 1999, ISBN-10: 0201379503, ISBN-13: 978-0201379501

[Burlisch05] Burlisch Roland, Stoer Josef : **Numerische Mathematik 2 - Eine Einführung unter Berücksichtigung von Vorlesungen von F.L.Bauer**, in: Fünfte Auflage, Springer Verlag, ISBN 3-540-23777-1

[Chapman07] Chapman Barbara, Jost Gabriele, Ruud van der Pas: **Using OpenMP - Portable Shared Memory Parallel Programming**, The MIT Press, 2007, ISBN-10: 0-262-53302-2, ISBN-13: 978-0-262-53302-7

[CLangCompiler] CLang Compiler Suite : http://clang.llvm.org/, Download am 23.06.2011

[CLN] CLN Webseite: http://www.ginac.de/CLN/Download, Download am 18.05.2007

[CodeAnalyst] AMD CodeAnalyst Performance Analyzer: http://developer.amd.com/tools/codeanalyst/pages/default.aspx, Download am 05.02.209

[Collatz66] Collatz, Lothar Dr.: **The numerical treatment of differential equations**, Springer Verlag, 1966, ISBN 3-540-03519-2

[cppcheck] Webseite des cppcheck-Projekts: http://cppcheck.sourceforge.net/, Download am 29.07.2011

[cTuning] cTuning Compiler Collection: http://ctuning.org/ctuning-cc, Download am 25.06.2011

[Denis06] Denis, St Tom; Rose, Greg: **BigNum Math: Implementing Cryptographic Multiple Precision Arithmetic**, Syngress Publishing Inc., 2006, ISBN 1597491128

[Deuflhard08] Deuflhard Peter, Hohmann Andreas : **Numerische Mathematik: Numerische Mathematik 1: Eine algorithmisch orientierte Einführung**, Gruyter, 2008, ISBN: 3110203545

[Echardt04] Eckhardt Bruno: **Chaos**, Fischer Kompakt Verlag, 2004, ISBN-3-596-15569-x

[EGM96] Webseite des Eearth Gravity Modells aus dem Jahr 1996: http://cddis.nasa.gov/926/egm96/egm96.html, Download 17.07.2011

[Engeln-Müllges05] Engeln-Müllgs Gisela, Niederdrenk Klaus, Wodica Reinhardt: **Numerik-Algorithmen**, Springer Verlag, 2005, ISSN 1439-5428, ISBN 3-540-62669-7

[Erler00] Thomas Erler, Dr.: **UML - Das Einsteigerseminar**, VMI - Verlag, 2000, ISBN 3-8287-1097-2

[Erler05] Erler Thomas, Ricken Michael: **UML 2 Ge-Packt**, mitp-Verlag/Bonn, 2005, ISBN-3-8266-1484-4

[Ettl10] Ettl Martin, Schneider Manfred, Hugentobler Urs: **Hochgenaue numerische Lösung von Bewegungsproblemen mit frei wählbarer Stellengenauigkeit - Simulation von GRACE – Bahnen und deren Abständen-**, Poster bei der FGS-Tagung in Bad Kötzting, 23.05.2010

[fbench] Walker John: **fbench - Trigonometry Intense Floating Point Benchmark** http://www.fourmilab.ch/fbench/fbench.html, Download am 27.07.2011

[Flechtner12] Flechtner Frank, Bettadpur Srinivas, Tapley Byron, Watkins Michael: **GRACE Mission Status**, EGU General Assembly, 2012, Wien

[Fursin10] Fursion Gregori, Temam Olivier: **Collective Optimization: A Practical Collaborative Approach**, in : Transactions on Architecture and Code Optimization, Dezember 2010, Volume 7, Number 4, pages 20-49

[Fursin11] Fursin Grigori, Kashnikov Yuriy, Memon Abdul Wahid, Chamski Zbigniew, Temam Olivier, Namolaru Mircea, Elad Yom-Tov, Bilha Mendelson, Ayal Zaks, Eric Courtois, Francois Bodin, Phil Barnard, Elton Ashton, Edwin Bonilla, John Thomson, Chris Williams, Michael O'Boyle: **MILEPOST GCC: machine learning enabled self-tuning compiler**, in : International Journal of Parallel Programming (IJPP), June 2011, Volume 39, Issue 3, pages 296-327

[Gamma01] Gamma Erich, Helm Richard, Johnson Ralph: **Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software**, 2001, Addison-Wesley ,ISBN-10: 3827318629, ISBN-13: 978-3827318626

[GiNAC] GiNAC Homepage: http://www.ginac.de, Download am 01.03.2011

[Gleixner86] Gleixner, H.: **Ein Beitrag zur Ephemeridenrechnung und Parameterschätzung im Erde-Mond-System**, Deutsche Geodätische Kommission, Reihe C, Nr. 319, München 1986

[GNUCompiler] GNU Compiler: http://gcc.gnu.org/onlinedocs/, Download am 22.06.2011

[Gough04] Gough Brian J.: **An Introduction to Gcc**, Network Therorie Ltd, 2004, ISBN-10: 9780954161798, ISBN-13: 978-0954161798, ASIN: 0954161793, auch Online verfügbar unter http://www.network-theory.co.uk/gcc/intro/, Download 02.04.2009

[GProf] GNU Profiler: http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html, Download am 24.06.2011

[Granston01] Granston Elena, Holler Anne: **Automatic Recommendation of Compiler Options**, in : Proceedings of the Workshop on Feedback-Directed and Dynamic Optimization, 2001, DOI 10.1.1.22.4333

[Gruber10] Gruber Thomas: **Vorlesungsunterlagen: Projekt Satellitenbahnen**, Technische Universität München, Juli 2010

[Hairer03] Hairer, Ernst; Lubich, Christian; Wanner, Gerhard (2003): **Geometric numerical integration illustrated by the Störmer/Verlet method**, Acta Numerica 12: 399–450. doi:10.1017/S0962492902000144

[Hairer96] Hairer, Ernst; Wanner, Gerhard: **Solving Ordinary Differential Equations II - Stiff and Differential-Algebraic Problems**, 2nd rev. ed. 1996. 3rd printing, 2010, XVI, 614 p. 137 illus., ISBN-978-3-540-60452-5

[Haneda05-1] Haneda M. , Knijnenburg P. M. W. , Wijshoff H. A. G.: **Optimizing General Purpose Compiler Optimization**, in ICS '05: Proceedings of the 2nd conference on Computing frontiers, May 2005

[Haneda05-2] Haneda M. , Knijnenburg P. M. W. , Wijshoff H. A. G.: **Generating new general compiler optimization settings**, in ICS '05: Proceedings of the 19th annual international conference on Supercomputing, June 2005

[Hanson97] Hanson David R. (1997): **C INTERFACES AND IMPLEMENTATIONS, Techniques for Creating Reusable Software**, ADDISON-WESLEY, ISBN 0-201-49841-3

[Hardy08] Hardy, Yorik; Tan Shi Kiat; Steeb Willi-Hans: **Computeralgebra with SymbolicC++**, World Scientific Printers, ISBN-13 978-981-283-360-0

[Herold] Herold Helmut: **Linux/Unix Systemprogrammierung**, Addison Wesley - Verlag

[Hoffmann08] Hoffmann, S.; Lienhart, R. : **OpenMP**, Springer Verlag, 2008, ISBN 978-3-540-73122-1

[Huckle06] Huckle Thomas, Schneider Stefan: **Numerische Methoden**, Springer - Verlag, 2.Auflage, ISBN-10 3-540-30316-2

[Hughes03] Hughes Cameron, Hughes Tracey: **Parallel and Distributed Programming Using C++**, Addison-Wesley, 2003, ISBN: 0-13-101376-9

[Hut95] Hut Piet, Makino Jun, McMillan Steve: **Building a better leapfrog**, The Astrophysical Journal, 443: L93-L96, 1995 April 20

[IEEE-754-1985] **IEEE Standard for Binary Floating-Point Arithmetic**, http://754r.ucbtest.org/standards/754.pdf

[Ince56] Ince E.L.: **Ordinary Differential Equations**, Dover Publications, Repirnt edition (June 1, 1956), ISBN-10: 0486603490, ISBN-13: 978-0486603490

[IntelCompiler] Intel®Compiler für Linux: http://software.intel.com/en-us/articles/intel-compilers/, Download am 19.01.2008

[IntelInspector] Intel®Inspector XE: http://software.intel.com/en-us/articles/intel-inspectorxe/, Download am 29.07.2011

[IntelVTune] Intel®VTune™Performance Analyzer with Intel®Thread Profiler : http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/, Download am 24.06.2011

[ISPC-Compiler] Intel®SPMD Program Compiler: http://ispc.github.com/index.html, Download am 22.06.2011

[Jeffrey07] Jeffrey Phil : **ABC Transporter Debacle**: http://xray0.princeton.edu/ phil/Facility/Guides/ABCtransporter.html, Download am 02.03.2011

[Jones05] Jones Tim M. : **Optimization in GCC**, The Linux Journal, Issue 131, März 2005, http://www.linuxjournal.com/article/7269?page=0,0, Download am 21.06.2011

[Jordan72] Jordan-Engeln Gisela Dr. rer. nat., Reutter Fritz Dr. rer. tech. : **Numerische Mathematik für Ingenieure**, Hochschultaschenbücher Band 104, Bibliographisches Institut/Mannheim/Wien/Zürich 1972

[Joshi01] Joshi Rajeev, Nelson Greg, Keith Randall : **Denali: a goal-directed superoptimizer**, SRC Research Report, 30 July 2001, http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-171.html, Download am 07.07.2011

[Kavanagh10] Kavanagh Etta: **Retraction Letter**, http://www5.in.tum.de/huckle/Retraction_, Download am 02.03.2011

[Kernighan90] Kernighan Brian W., Ritchie Dennis M.: **Programmieren in C**, 2.Auflage, 1990, Hanser - Verlag

[Kim00] Jeongrae Kim, Dissertation: **Simulation Study of A Low-Low Satellite-to-Satellite Tracking Mission**, University of Texas at Austin, 2000

[Kinoshita88] Kinoshita Hiroshi, Hiroshi Nakai : **Numerical integration methods in dynamical astronomy**, Celestial Mechanics and Dynamical Astronomy, 1988, Volume 45, Numbers 1-3, Pages 231-244

[Kinoshita98] Kinoshita Hiroshi, Hiroshi Nakai: **Numerical integration methods in dynamical astronomy** in CELESTIAL MECHANICS AND DYNAMICAL ASTRONOMY Volume 45, Numbers 1-3, 231-244, DOI: 10.1007/BF01229006

[Knuth98] Knuth E. Donald : **The Art of Computer Programming - Volume 2 - Seminumerical Algorithms**, Third Edition, Addison-Wesley 1998, ISBN 0-201-89684-2

[König03] König Daniel Dipl.-Ing., Seitz Kurt Dr.-Ing.: **Numerische Integration von Satellitenbahnen unter Berücksichtigung der Anisotropie des Gravitationsfeldes der Erde**, Schriftreihe des Studiengangs Geodäsie und Geoinformatik 2003 1, Universität Karlsruhe (TH)

[Ladd95] Ladd Scott Robert : **Genetic Algorithms in C++**, Hungry Minds Inc,U.S.; Auflage: Pap/Dsk (Dezember 1995), ISBN-10: 1558514597, ISBN-13: 978-1558514591

[LiDIA] LiDIA Webseite: http://www.cdc.informatik.tu-darmstadt.de/TI/LiDIA/, Download am 18.05.2007

[Lowe07] Lowe Derek: **Wrong, But Still Convincing**: http://pipeline.corante.com/-archives/-2007/-02/-27/, Download am 02.03.2011

[Mai11] Mai Enrico, Dr.: **Numerische Integration von Satellitenbahnen mittels Liereihen-Entwicklung, Habilitationsschrift**, Technische Universität Berlin, 181 Seiten, 2011, http://dgk.badw.de/fileadmin/docs/c-659.pdf

[Maple] MAPLE Webseite: http://www.maplesoft.com/products/maple/, Download am 02.03.2011

[MAPM] MAPM Webseite: http://www.tc.umn.edu/~ringx004/mapm-main.html, Download am 18.05.2007

[Martin09] Martin C. Robert: **Clean Code, Refactoring, Patterns, Testen und Techniken für sauberen Code**, mitp-Verlag, 2009, ISBN 978-3-8266-5548-7

[Mathematica] MATHEMATICA Webseite: http://www.wolfram.com/mathematica/, Download am 01.03.2011

[Mathews03] Mathews John: Module for Derivation of Numerical Differentiation Formulae - Computer Derivations of Numerical Differentiation Formulae (Classroom Notes) - http://math.fullerton.edu/mathews/n2003/NumericalDiffFormulaeMod.html, Download am 03.04.2008

[Maxima] MAXIMA Webseite: http://maxima.sourceforge.net/, Download am 01.03.2011

[Merali10] Zeeya Merali: **„Computational science: ...Error...why scientific programming does not compute.“**, Nature 467, 775-777 (2010) doi:10.1038/467775a, http://www.nature.com/news/2010/101013/full/467775a.html, Download am 18.02.2011

[Mikkola01] Mikkola Seppo, Aarseth Sverre: **A Time-Transformed Leapfrog Scheme. Integration of Few-Body-Systems with Large Mass Ratios**, in: Celestial Mechanics and Dynamical Astronomy 84, 2001, S. 343-354.

[MikTeX] MikTeX Homepage: http://www.miktex.org/, Download am 01.12.2005

[Miller06] Miller Greg: **A Scientist's Nightmare: Software Problem Leads to Five Retractions**, Science 22 December 2006 , Vol. 314 no. 5807 pp. 1856-1857, DOI: 10.1126/science.314.5807.1856

[Montenbruck05] Montenbruck Oliver, Gill Eberhardt : **Satellite Orbits**, Corrected 3rd printing 2005, Springer Verlag, ISBN-3-540-67280-X

[Montenbruck91] Montenbruck Oliver : **Ephemeridenrechnung und Bahnbestimmung geodätischer Satelliten mit Hilfe der Taylorreihenintegration**, Deutsche Geodätische Kommission, Reihe C, Dissertation, Heft Nr. 384, München 1991

[Morrison62] Morrison David : **Optimal mesh size in the numerical integration of an ordinary differential equation**, J. Association for Computing Machinery, Jan. 1962, page 98-103, ISSN 0004-5411, DOI 10.1145/321105.321115

[MPIStandard] Message Passing Interface Standard : http://www.mcs.anl.gov/research/projects/mpi/, Download am 24.06.2011

[MPL] Boost Metaprogramming Libray: http://www.boost.org/doc/, Download am 06.07.2011

[Muller10] Muller Jean-Michel, Brisebarre Nicolas, Florent de Dinechin, Jeannerod Claude-Pierre, Lefevre Vincent, Melquiond Guillaume, Revol Nathalie, Stehle Damien, Torres Serge: **Handbook of Floating Point Arithmetic**, Birkhäuser Verlag, 2010, ISBN-978-0-8176-4704-9

[Müller91] Müller J.: **Analyse von Laserentfernungsmessungen zum Mond im Rahmen einer post-Newton'schen Theorie.**, Deutsche Geodätische Kommission, Reihe C, Nr. 383, München 1991

[Neidhardt08] Neidhardt, A.; Ettl, M.; Dassing, R.; Hase, H.; Mühlbauer, M.; Plötz, Ch.; Sobarzo, S.; Herrera, C.; Alef, W.; Rottmann, H.: **The interface generator idl2rpc.pl and ideas for the automation of VLBI telescopes**; 8th RadioNet Engineering Workshop, Yebes,Spain, 24.11.2008

[Novillo05-1] Novillo Diego : **Performance Tuning with GCC, Part 1**, September 2005 http://www.redhat.com/magazine/011sep05/features/gcc/, Download am 22.06.2011

[Novillo05-2] Novillo Diego : **Performance tuning, Part 2: Analyzing performance problems**, October 2005 http://www.redhat.com/magazine/012oct05/features/gcc/, Download am 22.06.2011

[NTL] NTL Webseite: http://www.shoup.net/ntl/, Download am 18.05.2007

[NYU] Webseite der University of New York : http://www.nyu.edu/, Download am 22.05.2007

[Open64] Webseite des Open64-Compiler Projekts: http://www.open64.net/, Download am 23.06.2011

[OpenACC] Webseite des OpenACC Projekts: http://www.openacc-standard.org/, Download am 15.11.2011

[OpenMP] Open Multi-Processing Standard: http://www.openmp.org, Download am 02.06.2009

[OpenMPI] Open Message Passing Interface Standard: http://www.open-mpi.org/software/ompi/v1.4/, Download am 29.07.2011

[OpenWatcom] Webseite des Open Watcom Compiler Projekts: http://www.openwatcom.org/, Download am 23.06.2011

[PathScaleEKOCompiler] Webseite der PathScale EKO Compiler Suite Projekts: http://www.pathscale.com/ekopath-compiler-suite, Download am 20.06.2011

[PCC] Portable C Compiler (PCC): http://pcc.ludd.ltu.se/, Download am 23.06.2011

[Polyspace] Webseite der Firma Polyspace: Code Verification with Polyspace, http://www.mathworks.com/products/polyspace/index.html, Download am 25.07.2011

[Press07] Press H. William, Teukolsky A. Saul, Vetterling T. William, Flannery P. Brian: **Numerical Recipes in C, The Art of Scientific Computing**, Third Edition, Cambridge University Press, ISBN 978-0-521-88068-8

[Press92] Press H. William, Teukolsky A. Saul, Vetterling T. William, Flannery P. Brian: **Numerical Recipes in C, The Art of Scientific Computing, Second Edition**, Cambridge University Press, ISBN 0-521-43108-5

[Priest91] Douglas M. Priest : **Algorithms for Arbitrary Floating Point Arithmetic**, Department of Mathematics, University of California, Berkley, 1991

[Quinn04] Quinn, Michael Jay : **Parallele Programming in C with MPI and OpenMP**, McGraw Hill, 2004, ISBN 007-123265-6

[Rade97] Rade L., Westergren B. : **Springers Mathematische Formeln**, übersetzt und bearbeitet von P. Vachenauer, 2. Auflage, 1997, Springer Verlag, ISBN-10: 3540675051, ISBN-13: 978-3540675051

[Rauber08] Rauber, T.; Rünger, G. : **Multicore: Parallele Programmierung**, Springer Verlag, 2008, ISBN 978-3-540-73113-9

[Reichhoff99] Reichhoff, Burkard: Dissertation: **Verfeinerung und objektorientierte Implementierung eines Modells zur Nutzung von Lasermessungen zum Mond**; ISBN (Print) 3-7696-9550-X, 1999

[Reinsch00] Reinsch, M.W. : **A simple expression for the terms in the Baker–Campbell–Hausdorff series**, Journal of Mathematical Physics, Volume 41, Issue 4, MISCELLANEOUS TOPICS IN MATHEMATICAL METHODS, 41(4): 2434–2442, (2000). doi:10.1063/1.533250

[Ridders82] Ridders, C. J. F.: **Accurate Computation of F '(x) and F '(x)F''(x)**, 1982, Advances in Engineering Software, 4, 75–75., Online unter: http://www.scribd.com/doc/54615147/Accurate-Computing, Download am 02.08.2011

[RoseCompiler] Rose Compiler Webseite : http://www.rosecompiler.org/, Download am 28.06.2011

[Rosenband08] Rosenband T., Hume D. B., Schmidt P. O., Chou C. W, Brusch A., Lorini L., Oskay W. H., Drullinger R. E., Fortier T. M., Stalnaker J. E., Diddams S. A., Swann W. C., Newbury N. R., Itano W. M., Wineland D. J. , Bergquist J. C.: **Frequency Ratio of Al^+ and Hg^+ Single-Ion Optical Clocks; Metrology at the 17th Decimal Place**, Science 28 March 2008, Vol. 319 no. 5871 pp. 1808-1812, DOI: 10.1126/science.1154622

[Sauer11] Sauer Jürgen: **Neuronale Netze, Fuzzy Control-Systeme und Genetische Algorithmen**, Skriptum zur Vorlesung im WS 2010 / 2011, http://fbim.fh-regensburg.de/~saj39122/NN/index.html, Download am 22.03.2011

[Scheid88] Scheid Francis J.:**Schaum's Outline of Numerical Analysis (Schaum's Outlines)**, Mcgraw-Hill Professional, Auflage: 0002 (1. Oktober 1988), ISBN-10: 0070552215, ISBN-13: 978-0070552210

[Schneider92] Schneider Manfred: **Himmelsmechanik Bd. I - Grundlagen, Determinierung, Grundlagen, Determinierung, 3., völlig neu bearbeitete und erweiterte Auflage**, BI-Wiss.-Verlag, ISBN-3-411-15223-0

[Schneider93] Schneider Manfred: **Himmelsmechanik Bd. II - Systemmodelle**, 1993, BI-Wiss.-Verlag, ISBN-3-411-15981-2

[Schneider99] Schneider Manfred: **Himmelsmechanik Bd. IV - Theorie der Satellitenbewegung, Bahnbestimmung**, 1999, BI-Wiss.-Verlag, ISBN-3-8274-0484-3

[Schuster05] Schuster, H. G.; Just, W.: **Deterministic Chaos-An Introduction Fourth, Revised and Enlarged Edition**, 2005, WILEY-VCH Verlag GmbH, ISBN: 3527-40415-5

[Shirley07] Shirley Peter, Morley R. Keith: **Realistic Ray Tracing, Second Edition**, AK Peters, 2007, ISBN: 1568811985

[Sidi10] Sidi Avram: **Practical Extrapolation Methods, Theory and Applications**, Cambridge Monographs on Applied and Computational Mathematics, 2010, Online-ISBN 9780511546815

[Simeon07] Simeon, M.: **Vorlesungsskript Numerische Mathematik 2**, Technische Universität München SS 2007, Download am 22.08.2007

[Siminiski11] Sininski, Jan A.:**Master Thesis: Numerical Integration of Satellite Orbits across Shadow Boundaries**, Technische Universität München, 2011

[SolarisStudioExpress] Solaris Studio Express Compilersuite: http://www.oracle.com/technetwork/server-storage/solarisstudio/overview/index.html, Download 23.06.2011

[SPEC95] SPEC95: http://www.spec.org/benchmarks.html, Download 22.06.2011

[Speith07] Speith Roland : **Das klassische N-Körper Problem (Numerische Zeitintegratoren)**, http://www.tat.physik.uni-tuebingen.de/ speith/Projekt1/projekt1.pdf, Download am 22.10.2007

[Stephani95] Stephani Hans: **Theoretische Mechanik: Punkt und Kintinuumsmechanik** , Spektrum Akad. Verl. 1995, ISBN 3-86025-284-4

[Stroustrup04] Stroustrup Bjarne, Abrahams David, Gurtovoy Aleksey : **C++ Template Metaprogramming - Concepts, Tools, and Techniques from Boost and Beyond**, Addison-Wesley, ISBN 0-321-22725-5

[Sutter05] Sutter Herb, Alexandrescu Andrei : **C++ Coding Standards -101 Rules, Guidlines, and best Practices**, Pearson Education, 2005, ISBN 0-321-11358-6

[Sutter09] Sutter Herb : **Exceptional C++ - 47 Engineering Puzzles, Programming Problems, and Solutions**, Addison Wesley, 2009, ISBN 0-201-61562-2

[SymbolicC++] Symbolic C++ Homepage : http://issc.uj.ac.za/symbolic/symbolic.html, Download am 04.08.2010

[Taylor05] Taylor, John R.: **Classical Mechanics**, University Science Books, 2005, Edwards Brothers, Inc. ISBN 189138922X

[TCC] Tiny C Compiler (TCC): http://bellard.org/tcc/, Download 23.06.2011

[ttmath] ttmath-Bibliothek (C++): http://www.ttmath.org/ttmath, Download am 30.11.2010

[TUD] Webseite der Technische Universität Darmstadt : http://www.tu-darmstadt.de/, Download am 22.05.2007

[UBUNTU] Webseite von Ubuntu Linux: http://www.ubuntu.org, Download am 23.05.2007

[Unruh94] Unruh Erwin: **Prime number computation**, 1994, ANSI X3J16-94-0075/ISO WG21-462

[UOM] Webseite der University of Minnesota : http://www1.umn.edu/twincities/index.php, Download am 23.05.2007

[Vandevoorde08] Vandevoorde David, Josuttis Nicolai M. : **C++ Templates - The Complete Guide**, Pearson Education, 2008, ISBN 0-201-73484-2

[Verdonk04] Verdonk B., Cuyt. A.: **Reliable floating-point arithmetic - the need for testing** , University of Antwerp, Wuppertal, October 27, 2004, ftp://ftp.win.ua.ac.be/pub/cant/talks/wuppertal1.pdf, Download am 05.08.2011

[Verlet67] Verlet Loup: **Compute 'Experiments' on Classical Fluids**, I. Thermodynamical Properties of Lennard-Jones Molecules, in : Physical Review, 159(1):98-103, 1967

[VonHagen08] Von Hagen William : **The Definitive Guide to GCC**, APress Verlag, 2008, ISBN-10: 9781590595855, ISBN-13: 978-1590595855, ASIN: 1590595858

[Wanner68] Wanner Gerhard: **Integration Gewöhnlicher Differentialgleichungen - Lie Reihen (mit Programmen) und Runge-Kutta Methoden**, Hochschultaschenbücher, 831/831a

[Watkins10] Watkins (JPL) M., Gross (JPL) M., Tapley (UTCSR) B., Bettadpur S. (UTCSR), F. Flechtner, B. Doll, J. Munder, C. Reigber, J.-C Raimondo: **GRACE Follow-On Mission Status**, 2010

[Wolf06] Wolf Jürgen : **C von A bis Z Das umfassende Handbuch für Linux, Unix und Windows**, 2. aktualisierte und erweiterte Auflage, 2006, Galileo Computing, ISBN 3-89842-643-2

[wxFormBuilder] Webseite des Projekts: http://wxformbuilder.org/, Download am 12.10.2011

[wxWidgets] Webseite des Projekts: http://www.wxwidgets.org, Download am 12.10.2011

[x86Open64Compiler] x86 Open64 Compiler Suite : http://developer.amd.com/tools/open64/Pages/default.aspx, Download am 23.06.2011

[Yoshida90] Yoshida Haruo: **Construction of higher order symplectic integrators**, NationalAstronomical Observatory, Mitaka, Tokyo 181,Japan, 1990, Phys. Lett. A 150: 262. doi:10.1016/0375-9601(90)90092-3

[Zimmermann98] Zimmerann Paul (Paul.Zimmermann@inria.fr): **Comparison of three public-domain multiprecision libraries: BigNum, Gmp and Pari**, February 12, 1998, Download unter : http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.36.6864 (Download am 25.11.2011)